AMIGA

Jorgo Schimanski

grafik in Assembler



Auf 3½"-Diskette enthalten:
Sämtliche Beispielprogramme des Buches
und nützliche Hilfsprogramme







Jorgo Schimanski

grafik in Assembler

Heim Verlag

Auf 3½"-Diskette enthalten:
Sämtliche Beispielprogramme des Buches
und nützliche Hilfsprogramme

Grafik in Assembler auf dem Amiga
Jorgo Schimanski
— 1. Auflage — Darmstadt: Heim, 1990
ISBN 3-923250-90-8

beim Heim -Verlag · Organisation + Datentechnik
Heidelberger Landstr. 194 · 6100 Darmstadt
Telefon 06151-56057

Druck: Druckerei der Heim OHG, 6100 Darmstadt.

Inhaltsverzeichnis

1. Kapitel

Primitive Grafikprogrammierung

- Bitmaps - Die Grafikspeicher
- Der Rasterport
- View - Der Verwalter
- ViewPort - Das Grafikfenster
- Colormap - Der Farbenmischer
- RasInfo - Die Bildverbindung
- Wie man Strukturen initialisiert!
- Das erste eigene Display
- Grafikmodi:
  - Hold and Modify (HAM)
  - Hires
  - Dualplayfield
  - Extra-Halfbrite
  - Scrolling
  - Double-Buffering - Blitzschnelle Grafik

1. Primitive Grafikprogrammierung

Die primitive Grafikprogrammierung ist die unterste Stufe der Grafikprogramierung. Es wird hier auf die Routinen der 'graphics.library' zurückgegriffen. Diese Routinen stellen dann den kontakt zur Hardware her. Dadurch ist es möglich, noch relativ komfortabel irgendwelche grafischen Effekte zu erzeugen, ohne sich mit der komplizierten Hardware herumschlagen zu müssen. Leider sind die Routinen des Systems nicht ganz so schnell, als wenn man sie selbst programmieren würde. Dieses ist aber auf die Programmiersprache zurückzuführen, in der das Betriebssystem geschrieben wurde. Das System wurde unverständlicherweise in 'C' geschrieben. Jedoch kann man sich somit unwahrscheinlich viel Programmierarbeit ersparen. Außerdem programmieren Sie ja in Assembler und haben somit schon wieder ein kleinen Vorteil gegenüber den C-Programmierern.

1.1 Bitmaps - Die Grafikspeicher

Um auf dem Amiga Grafik bzw. Animationen zu programmieren benötigt man als Grundelement den Speicher, in der die Grafik dargestellt wird. Die sogenannte Bitmap! Damit das System weiß wo sich nun die Grafik befindet und wie groß sie ist, gibt es eine Bitmapstruktur. Eine Struktur ist nichts anderes, als eine Tabelle von einer bestimmten Länge, die an das System übergeben wird und in der alle nötigen Parameter für die Bitmap zu finden sind. Das System arbeitet fast ausschließlich über solche Strukturen, weil es halt die schnellste Möglichkeit ist, eine große Anzahl von Parametern zu übergeben. Hier nun der Aufbau der Bitmap-Struktur:

'Bitmap'-Struktur: (Länge = 40 Bytes)

Offset:	Typ	Bezeichnung	Beschreibung
000	Word	BytePerRow	Bytes pro Display-Zeile
002	Word	Rows	Anzahl der Display-Zeilen
004	Byte	Flags	System-Flags
005	Byte	Depth	Anzahl der Bitmaps (Tiefe)
006	Word	Pad	unbenutzt
008	Long	PlanePtr1	Zeiger auf 1. Bitmap (02 Farben)
012	Long	PlanePtr2	Zeiger auf 2. Bitmap (04 Farben)
016	Long	PlanePtr3	Zeiger auf 3. Bitmap (08 Farben)
020	Long	PlanePtr4	Zeiger auf 4. Bitmap (16 Farben)
024	Long	PlanePtr5	Zeiger auf 5. Bitmap (32 Farben)
028	Long	PlanePtr6	Zeiger auf 6. Bitmap (HAM)
032	Long	PlanePtr7	Zeiger auf 7. Bitmap (unbenutzt)
036	Long	PlanePtr8	Zeiger auf 8. Bitmap (unbenutzt)
040		END	Ende der 'Bitmap'-Struktur

Die ersten 8 Bytes sind Systemvariablen und werden automatisch vom System initialisiert. Nur die PlanePtrX müssen von Ihnen eingetragen werden.

Nehmen wir mal an, wir wollen einen Screen der 320 Punkte breit und 256 Punkte hoch ist erzeugen. Und dieser Screen soll 32 Farben besitzen. Also benötigen wir 5 Bitplanepointer. Jede einzelne Bitmap ist 320 X 256 Bits (Pixel) groß, also 320/8*256 = 10240 By-

tes und das gesamte Display benötigt einen Speicher von 10240*5 = 51200 Bytes. Diesen Speicher reservieren wir am besten mit der Routine 'AllocMem ()' aus der Exec-Bibliothek. Nach dem wir nun den Speicher für die Bitmaps erstellt haben, legen wir die 'Bitmapstruktur' an, initialisieren also die Plane-pointers! Jetzt endlich können wir die Bitmapstruktur vom System mit den fehlenden Parametern füllen lassen. Dies erledigt die Routine 'Init-BitMap ()' aus der Graphics-Bibliothek.

Um nun einen Punkt in einer bestimmten Farbe zu erzeugen, hängt ganz davon ab, in welcher Bitmap bzw. in welchen Bitmaps wir einen Punkt setzen. Gehen wir erstmal von einer Bitmap aus und stellen sie sich vor, diese ist genauso groß, wie ein kariertes Blatt Papier und auf diesem Blatt befinden sich 320 Kästchen in einer Reihe und davon gibt es 256 Reihen. In jedes Kästchen, wo Sie nun ein Loch reinstechen wird ein Punkt,in der gleichen Position auf dem Bildschirm dargestellt. Man kann mit einer Bitmap also nur die Zustände 0 = kein Loch (kein Punkt) oder 1 = Loch (Punkt) darstellen. Man besitzt einen Wertebereich von 0 - 1. Diese Nummern sind gleichzeitig die Farbregisternummern aus denen die Farbe für einen Punkt entnommen werden. Um nun mehr Zustände darstellen zu können, nimmt man einfach noch eine Bitmap dazu. Diese Bitmaps werden vom System automatisch übereinander gelegt. Also stellen Sie sich vor, Sie legen zwei karierte Blätter genau übereinander. Jetzt können Sie einen Punkt schon in vier Farben darstellen, welches Farbregister Sie ansteuern, hängt ganz davon ab, wo Sie nun die Löcher reinstechen. Folgende Tabelle soll dies nun verdeutlichen:

Blatt (Bitmap) 1:	**Blatt (Bitmap) 2:**	**Bitkombination:**	**Farbregister (Wert):**
kein Loch	kein Loch	00	00 (Hintergrundfarbe)
Loch	kein Loch	01	01
kein Loch	Loch	10	02
Loch	Loch	11	03

Bei drei Bitmaps ist die Bitkombination natürlich dreistellig usw. Je mehr Bitkombinationen, desto mehr Werte lassen sich darstellen, desto mehr Farben stehen einem zur Verfügung.

Hier nun ein kurzes Beispielprogramm, was auf dem Workbench-Screen einen kurzen Balken zeichnet:

```
;-------------------- Programmname : BitMap ---------------------

start:  move.l #intname,a1   ; Zeiger auf Library-Name
        clr.l d0             ; Versionsnummer = 0
        move.l 4,a6          ; Execbasisadresse nach A6
        jsr -552(a6)         ; OpenLibrary-Routine aufrufen
        move.l d0,intbase    ; Adresse der Intuitionbasis speichern
        tst.l d0             ; auf Error
        beq exit             ; testen

        move.l d0,a0         ; Intuitionbasis nach A0
        move.l 52(a0),a0     ; Zeiger auf aktuelles Fenster
        move.l 46(a0),a0     ; Zeiger auf Workbench-Screenadresse
        add.l #184,a0        ; A0 = Anfangsadresse der Bitmapstruktur
        move.l 8(a0),a1      ; A1=Zeiger auf erste Bitmap vom WBScreen
        move.l 12(a0),a2     ; A2=Zeiger auf zweite Bitmap vom WBScreen

        move.l #640/8,d0             ; Länge einer Zeile in Bytes
        move.l #80/8+55*640/8,d1     ; X,Y Position des Balkens
        add.l d1,a1          ; Anfangsadresse in der Bitmap1 errechnen
        add.l d1,a2          ; Anfangsadresse in der Bitmap2 errechnen
        clr.w d2             ; Zähler für Länge des Balkens
        clr.w d3

draw_loop:  move.b #$ff,(a1,d3)   ; 8 Punkte in einer Zeile zeichnen
                                  ; (Bitmap 1)
            move.b #$ff,(a2,d3)   ; 8 Punkte in einer Zeile zeichnen
                                  ; (Bitmap 2)
```

```
            add.w #1,d2            ; Zeilenzähler um 1 erhöhen
            move.w d0,d3           ; Länge einer Zeile nach D3
            mulu d2,d3             ; mit Zeilenposition multiplizieren
            cmp.w #50,d2           ; schon 50 Zeilen ?
            bne draw_loop          ; wenn nicht, dann weiter

exit:       move.l intbase,a1      ; Intuitionbasis nach A1
            cmp.l #0,a1            ; auf Error
            bne exit1              ; testen
            rts                    ; wenn Error, dann Programmende

exit1:      move.l 4,a6            ; Execbasisadresse nach A6
            jsr -414(a6)           ; CloseLibrary-Routine aufrufen
            rts                    ; Programmende

;--------------------------- Parameter ----------------------------

intname: dc.b "intuition.library",0
even
intbase: dc.l 0

;------------------------- Programmende ---------------------------
```

Dieses Programm ist so gut dokumentiert, daß jede weitere Erläuterung überflüssig ist.

1.2 Der Rasterport

Fast alle Routinen des Betriebsystems benötigen eine Raster-Port-Struktur. In dieser Struktur werden nähmlich alle wichtigen Parameter für die Grafikprogrammierung gespeichert. Alle Unterstrukturen wie z. B. die Bitmapstruktur, werden mit dem Raster-Port verbunden.

In dem RasterPort stehen unteranderem Dinge, wie Farbe des Vordergrundstiftes, Größe und Breite des Screens bzw. Windows, welcher Zeichensatz (Font) gerade verwendet wird usw.

Wie schon kurz erwähnt, besitzt jedes Window und jeder Screen seine eigene RasterPort-Struktur. Mit anderen Worten also, der RasterPort verwaltet die Zeichenebenen auf dem Amiga! Die RasterPort-Struktur ist wie folgt aufgebaut:

'RastPort'-Struktur (Länge = 100 Bytes)

Offset	Typ	Bezeichnung	Beschreibung
000	Long	Layer	Zeiger auf 'Layer'-Struktur
004	Long	BitMap	Zeiger auf 'BitMap'-Struktur
008	Long	AreaPtrn	Zeiger auf das AreaFill-Muster
012	Long	TmpRas	Zeiger auf 'TmpRas'-Struktur
016	Long	AreaInfo	Zeiger auf 'AreaInfo'-Struktur
020	Long	GelsInfo	Zeiger auf 'GelsInfo'-Struktur
024	Byte	Mask	Schreibmaske (wieviel Bitmaps zugelassen sind)
025	Byte	FgPen	Farbregisternummer für Vordergrundfarbe
026	Byte	BgPen	Farbregisternummer der Hintergrundfarbe
027	Byte	AOlPen	Farbregisternummer der AreaFill-Outlinefarbe

028	Byte	DrawMode	ZeichenModi (0=Jam1, 1=Jam2, 2=Complement, 3=invers) für Rou tine 'SetDrMd ()'
029	Byte	AreaPtSz	Höhe des AreaFill-Musters (2^n Words))
030	Byte	LinePatCNT	unbenutzt
031	Byte	Dummy	Line Draw Pattern Preshift
032	Word	Flags	verschiedene Kontrollbits (First Dot,OneDot etc.)
034	Word	LinePtrn	16 Bits für Linienmuster
036	Word	CP_X	x-Position des Grafikcursors
038	Word	CP_Y	y-Position des Grafikcursors
040	Byte	Minterms(8)	8 x 1 Byte für Minterms (Funkti on unbekannt)
048	Word	PenWidth	Cursorbreite
050	Word	PenHeight	Cursorhöhe
052	Long	TextFont	Zeiger auf 'TextFont'-Struktur
056	Byte	AlgoStyle	Zeichensatzmodus (Style-Flag)
057	Byte	TxFlags	textspezifische Flags
058	Word	TxHeight	Höhe des Zeichensatzes
060	Word	TxWidth	durchschnittliche Zeichenbreite
062	Word	TxBaseLine	Texthöhe ohne Unterlängen (für 'Text'-Routine - Wert zu Y-Positi on addieren = genaue Y-Pos.)
064	Word	TxSpacing	Zeichenabstand
066	Long	RP-User	Zeiger auf evtuelle Userdaten (un wichtig!)
070	Word	7 Words	reserviert
084	Long	2 Longs	reserviert
092	Byte	8 Bytes	reserviert
100		END	Ende der Datenstruktur

Mit der Routine 'InitRastPort ()' aus der Graphics-Biliothek wird diese Struktur initialisiert!

Es folgt nun eine Beschreibung der wichtigsten Teile des Raster-Ports:

Offset 4: BitMap

Hier befindet sich die Adresse der Bitmapstruktur. Wie eine Bit-Map-Struktur aufgebaut und was eine Bitmap überhaupt ist, wurde ja bereits schon oben weiter erläutert. Wenn man allerdings eine eigene BitMap-Struktur angelegt hat, muß man die Adresse hier speichern.

z. B. `move.l #bitmap,rastport+4`

Offset 20: GelsInfo

Wenn man das Animationssystem der Graphics-Bibliothek benutzen, also BOBs und Sprites darstellen möchte, muß man als erstes das Animationssystem initialisieren. Dieses geschicht über die GelsInfo-Struktur. Danach muß dann die Adresse dieser Struktur hier gespeichert werden.

z. B. `move.l #gelsinfo,rastport+20`

Die Struktur wird im Kapitel "Das Animationssystem" beschrieben.

Offset 24: Mask

Hier steht die Anzahl der Bitmaps, welche Beschrieben werden können. Normalerweise steht hier der Wert 255. Durch ändern des Wertes z. B. auf Null, wird nichts mehr auf dem Screen gezeichnet.

Offset 36 und 38: CP_X und CP_Y

Diese beiden Wörter beinhalten die Position des Grafikcursors. Man kann diese Werte mit der Routine 'Move ()' aus der Graphics-Biblitothek setzen oder direkt hier hinein 'Poken', was zum selben Ergebnis führt, jedoch um einiges schneller ist.

Offset 56: AlgoStyle

Wie Ihnen bestimmt schon aufgefallen ist, kann der Amiga Zeichensätze in verschiedenen Formen auf den Bildschirm bringen (schräg etc.). In welcher Form der Zeichensatz auf dem Screen erscheinen soll, hängt ganz von dem Wert ab, der hier gespeichert ist:

Wert:	Form:
0	Normal
1	Underline (unterstrichen)
2	Bold (Fettdruck)
4	Italic (schräg - kursiv)

Es ist auch möglich mehrere Formen gleichzeitig darzustellen. Dazu braucht man nur die gewünschten Werte zusammenzuaddieren. Wie diese Werte gesetzt werden, wird im Font-Kapitel beschrieben.

Das war nun eine kurze Beschreibung der wichtigsten Parameter der RastPort-Struktur. Die noch fehlenden Parameter, sofern sie von Bedeutung sind, werden in anderen Kapiteln erläutert die natürlich auch mit Beispielprogrammen versehen sind.

1.3 View - Der Verwalter

Da ein Bild 50 mal (50 Hz) in der Sekunde aufgebaut wird, muß der Programmierer dafür sorgen, daß die Speicherstellen, in denen die Adressen der Bitmaps stehen, nach 1/50 Sekunde wieder auf den Anfangswert gesetzt werden. Denn das Bild wird Pixelweise aus dem Speicher, über den Rasterstrahl, auf den Bildschirm übertragen. Damit der Rasterstrahl weiß, an welcher Stelle er sich gerade im Speicher befindet, werden die Register, in denen die Adressen der Bitmaps gespeichert sind, intern raufgezählt. Wenn nun das Bild komplett auf dem Bildschirm übertragen wurde, würden die Register weiter raufgezählt werden, was zur Folge hätte, daß immer ein anderes Bild auf dem Bildschirm erscheint. Es gibt nur zwei Möglichkeiten dies zu bewerkstelligen; entweder im Vertikal-Blank-Interrupt oder über den Copper.

Das System benutzt dazu den Copper, weil es die einfachste und schnellste Art ist. Der Copper (Coprozessor) benötigt dazu eine sogenannte Copperlist, in der Befehle enthalten sind, die der Copper abarbeitet.

Um nun sein eigenes Display, mit Hilfe einer Copperlist, über die Systemroutinen zu programmieren und nicht direkt über die Hardware, exestieren folgende Strukturen:

1. 'View'-Struktur
2. 'ViewPort'-Struktur

Die 'View'-Struktur verwaltet das gesammte Display auf dem Amiga, es ist immer nur eine 'View'-Struktur im System, sie stellt die wichtigste Verbindung zwischen System und User dar.

Der Viewport, der mittels der 'ViewPort'-Struktur beschrieben wird, ist das 'Fenster', durch das man seine Bitmap auf dem Monitor sehen kann. Man kann soviele ViewPortstrukturen (Screens)

erzeugen wie man möchte, sie werden alle durch die 'View'-Struktur miteinander verbunden. Mit Hilfe der Routine 'MakeVPort ()' werden dann die einzelnen Copperlisten für Farbe etc. erzeugt bzw. neu berechnet. Mit der Routine 'MrgCop ()' werden dann die einzelnen Copperlisten zu einer zusammengefaßt. Und die Routine 'LoadView ()' startet dann die neue Copperliste über die Hardware.

Es folgt nun eine Beschreibung der View-Struktur:

'View'-Struktur: (Länge = 18 Bytes)

Offset	Typ	Bezeichnung	Beschreibung
000	Long	ViewPort	Zeiger auf erste 'View Port'-Struktur
004	Long	LOFCprList	Zeiger auf LongFrame-Copperli ste wird mit 'MrgCop ()' erzeugt
008	Long	SHFCprList	Zeiger auf ShortFrame-Copperli ste
012	Word	DyOffset	y-Position des Displays (wird vom System init.)
014	Word	DxOffset	x-Position des Displays (wird vom System init.)
016	Word	Modes	Modus-Flag (Inerlace-Flag hier setzen, falls gewünscht)
018		END	Ende der 'View'-Struktur

Wenn man nicht auf Screens der Intuitions-Bibliothek zurückgreifen möchte, sondern ein absolut eigenes Display erzeugen will, dann erst muß man eine View-Struktur anlegen.

Diese Struktur muß nur mit der angelegten ViewPort-Struktur verbunden werden (Offset 0). Falls ein Interlace Display gewünscht wird, muß bereits hier schon, in 'Modes' das Interlace-Flag gesetzt werden (Bitbelegung siehe ViewPort-Struktur).

Solche Displays haben allerdings einen Nachteil bzw. Vorteil; es können keine Menüs mehr programmiert werden und das Display kann auch nicht mehr mit der Maus hoch oder runter verschoben werden.

1.4 ViewPort - Das Grafikfenster

Wie schon erwähnt, ist der ViewPort das Grafikfenster, durch den man seinen Screen auf dem Bildschirm sieht. Jeder Screen besitzt seinen eigenen ViewPort. Dadurch ist es nähmlich möglich mehrere Screens zu überlappen und wenn man mit der Maus seinen Screen hoch bzw. runter bewegt, werden die Copperlisten für diesen ViewPort (Screen) neuberechnet und mit der View-Struktur verbunden. Danach wird dann die neue Copperliste für den VIEW erzeugt und über die Hardware gestartet. Jetzt erst erscheint der Screen an seiner neuen Position.dies erledigt das System im Rasterinterrupt, deswegen kann man die Screens auch nicht besonders schnell bewegen, dafür aber fließend.

Datenstruktur des ViewPort:

'ViewPort'-Struktur: (Länge = 40 Bytes)

Offset	**Typ**	**Bezeichnung**	**Beschreibung**
000	Long	Next	Zeiger auf nächste 'View Port'-Struktur
004	Long	ColorMap	Zeiger auf 'ColorMap'-Struktur
008	Long	DspIns	Zeiger auf Display-Copperlist struktur
012	Long	SprIns	Zeiger auf Sprite-Copperliststruk turl
016	Long	ClrIns	Zeiger auf Sprite-Copperliststruk tur2
020	Long	UCopList	Zeiger auf 'UCopList'-Struktur (für User)
024	Word	DWidth	Breite des Displays (Screen) in Pi xel
026	Word	DHeight	Höhe des Displays (Screen) in Zeilen

028	Word	DxOffset	X-Pos. des Displays (Screen)
030	Word	DyOffset	Y-Pos. des Displays (Screen)
032	Word	Modes	ViewPort-Modi:
			Wert $0000 = Normal alle Bits =0
			Wert $0002 = GenlockVideo Bit 1=1
			Wert $0004 = Interlace Bit 2=1
			Wert $0040 = PFBA Bit 6=1
			Wert $0080 = ExtraHalfbrite Bit 7=1
			Wert $0100 = GenlockAudio Bit 8=1
			Wert $0400 = DualPlayField Bit 10=1
			Wert $0800 = Hold + Modify Bit 11=1
			Wert $2000 = VP-Hide Bit 13=1
			Wert $4000 = Sprites Bit 14=1
			Wert $8000 = Hires Bit 15=1
034	Word	reserved	reserviert
036	Long	RasInfo	Zeiger auf 'RasInfo'-Struktur
040		END	Ende der 'ViewPort'-Struktur

Nach dem die Routine 'InitVPort ()' aufgerufen wurde, müssen nachträglich DWidth, DHeight, und Modes mit den gewünschten Werten gefällt werden. Wenn man bei 'Modes' das 'VP-Hide' Bit setzt, wird dieser Viewport in den Hintergrund gedrängt und somit nicht mehr vom System bearbeitet. So kann man z. B. die Abfrage der rechten/linken Maustaste abstellen. Wenn man in seinem ViewPort Sprites verwenden will, muß man das 'Sprites' Bit in 'Modes' setzen. Die restlichen Modis werden im Kapitel 'Grafik-Modis' beschrieben.

Des weiteren sind in der 'ViewPort'-Struktur noch zwei Zeiger, welche einmal auf die 'ColorMap'-Struktur und auf die 'RasInfo'-Struktur zeigen. In der ColorMap-Struktur stehen natürlich die Farben für den dazugehörigen ViewPort. Die 'RasInfo'-Struktur ist zuständig für die Bitmapverwaltung. Mit ihr ist es möglich Scrolling und DoubleBuffering-Bobs zu programmieren. Die Anfangsadressen dieser beiden Strukturen müssen, nachdem sie von Ihnen angelegt wurden, hier eingetragen werden.

1.5 Colormap - Der Farbenmischer

Bis jetzt sprachen darüber, wie man eine Bitmap erzeugt, sie initialisiert und wie man Punkte setzt bzw. löscht. Jedoch wie trägt man die Farbwerte der gewünschten Farben in die entsprechenden Register ein?

Dazu exestiert, wie sollte es auch anders sein, eine Struktur - die Color- Map-Struktur. In ihr sind die Farbwerte und die Anzahl der Farben gespeichert und ist wie folgt aufgebaut:

'ColorMap'-Struktur: (Länge = 8 Bytes)

Offset	Typ	Bezeichnung	Beschreibung
000	Byte	Flag	System-Flag
001	Byte	Typ	System
002	Word	Count	Anzahl Farben (max. 32)
004	Long	ColorTable	Zeiger auf einen Puffer mit 32 Words (64 Bytes) in dem die Farb werte stehen
008		END	Ende der 'ColorMap'-Struktur

Unter 'Count' wird die Anzahl der Farben gespeichert. Diese hängt, wie Sie bereits wissen, von der Anzahl der verwendeten Bitmaps ab. Den 'ColorTable'- Zeiger müssen Sie selbst eintragen. Dort wird die Anfangsadresse von einem 32 Word großen Array gespeichert. Dieses Array beinhaltet nähmlich die maximal 32 möglichen Farbwerte.

```
move.l #colortable,colormap+4
rts
;
colortable: blk.w 32,0          ; 32 Word-Array
```

Obiges kleines Assemblerlisting zeigt, wie man diesen Array programmiert.

Die Hardwareregister, welche die Farben aufnehmen, sind jeweils 1 Word Groß. Ein Word besitzt ja bekanntlich 16 Bits, jedoch werden für die Farben nur die unteren 12 Bits genutzt. Dabei enthalten die Bits 11-8 den Rot-, die Bits 7-4 den Grün- und die Bits 3-0 den Blauanteil einer Farbe.

So nun haben wir die Farben initialisiert. Aber woher weiß das System, wo sich die Colormap-Struktur im Speicher befindet? Dazu müssen Sie noch die Anfangsadresse der Colormap in die ViewPort-Struktur eintragen (Offset 4).

z. B. `move.l #colormap,viewport+4`

Allerdings werden die Farben nicht sofort sichtbar. Das hängt daran, daß die Copperliste, die die Farben verwaltet, erst immer auf den neusten Stand gebracht werden muß. Dieses erreicht man durch den Aufruf der drei Routinen 'MakeVPort ()', 'MrgCop ()' und 'LoadView ()' (bei Intuition-Screens reicht 'RemakeDisplay ()').

Man kann sich die Colormap-Struktur aber auch vom System anlegen lassen. Diese ColorMap wird mit den Farben des aktuellen Screens geladen. Das System bietet einem dazu zwei Routinen. Eine die die Colormap-Struktur anlegt und eine die sie wieder freigibt.

```
Routine  :  GetColorMap (entries) (D0)
Library  :  graphics.library
Offset   :  -570 = -$23a
Parameter:  D0 = Anzahl Farben (max. 32 Farben)
Rückgabe :  in D0 = Anfangsadresse der vom System angelegten 'Color-
            Map'-Struktur
```

Erklärung: Diese Routine erzeugt eine ColorMap-Struktur, deren Adresse in D0 zurückgegeben wird. Wenn eine Null zurückgegeben wird, dann konnte nicht genug Speicher für diese Routine angelegt werden. Diese vom System angelegte Struktur muß man dann nur noch mit der `View-Port`-Struktur verbinden (Offset 4).

Routine : FreeColorMap (ColorMap) (A0)
Library : graphics.library
Offset : -576 = -$240
Parameter: A0 = Anfangsadresse der ColorMap-Struktur, die mit der Routine `GetColorMap ()` angelegt wurde
Rückgabe : keine
Erklärung: Diese Routine gibt den Speicher wieder frei, der mit `GetColorMap ()` für die `ColorMap`-Struktur angelegt wurde.

Das System bietet einem noch die Möglichkeit über ein paar Routinen, mit der Colormap ein wenig herum zu manipulieren:

Routine : LoadRGB4 (ViewPort,Colors,Count) (A0,A1,D0)
Library : graphics.library
Offset : -192 = -$c0
Parameter: A0 = Anfangsadresse der zu diesem Display gehörenden ViewPort-Struktur
A1 = Anfangsadresse vom einem 32 Word groooßen Array, in denen sich die Farbwerte befinden
D0 = Anzahl der Farbregister, welche mit neuen Farben gefüllt werden soll
Rückgabe : keine
Erklärung: Diese Routine lädt in die ColorMap des angegebenen View-Ports die gewünschten Farbwerte, welche sich in dem Array befinden. Es müssen danach aber erst die Copperlisten neu berechnet werden, damit der Farbwechsel sichtbar wird.

Routine : SetRGB4 (ViewPort,Register,rot,gelb,blau) (A0, D0, D1, D2, D3)
Library : graphics.library
Offset : -288 = -$120
Parameter: A0 = Anfangsadresse der zu diesem Display gehörenden ViewPort-Struktur
D0 = Farbregisternummer, in dem die Farbänderung vorgenommen werden soll (0-31)
D1 = Rotanteil der Farbe (Wert von 0-15)
D2 = Gelbanteil der Farbe (Wert von 0-15)
D3 = Blauanteil der Farbe (Wert von 0-15)
Rückgabe : keine
Erklärung: Diese Routine lädt das angegebene Farbregister des entsprechenden ViewPorts mit dem gewünschten Wert. Die Farbänderung wird sofort sichtbar, die Copperlisten brauchen also nicht neu berechnet werden.

Routine : GetRGB4 (ColorMap,entry) (A0,D0)
Library : graphics.library
Offset : -582 = -$246
Parameter: A0 = Anfangsadresse der zu diesem Display gehörenden ColorMap-Struktur
D0 = Farbregisternummer, aus dem der Farbwert entnommen werden soll (0-31)
Rückgabe : in D0 = Farbwert des angegebenen Farbregisters
Erklärung: Diese Routine gibt den Farbwert des gewünschten Farbregisters in D0 zurück.

1.6 RasInfo - Die Bildverbindung

Jetzt fehlt uns nur noch der RasInfo-Zeiger aus der ViewPort-Struktur. Denn was nützt uns eine Bitmap, die irgend wo im Speicher liegt. Woher soll der ViewPort wissen, wo die BitMap im Speicher liegt. Und ab welcher Position die BitMaps auf dem Bildschirm erscheinen sollen. Durch die Positionierung ist es nähmlich möglich, bei Übergroßen Bitmaps nur einen Teil anzeigen zu lassen.

Auch hierfür exestiert wiederum eine Struktur, welche sich RasInfo nennt. Die RasInfo-Struktur ist zuständig für die Bitmapverwaltung. Mit dem RasInfo ist es auch möglich, Scrolling und doppelt gepufferte BOBs zu programmieren.

Die RasInfo-Struktur als solches, ist eigentlich absolut unwichtig. Sic fungiert nur als Bindeglied zwischen BitMap- und View-Port-Struktur. Hier nun der Aufbau der RasInfo-Struktur:

'RasInfo'-Struktur: (Länge = 12 Bytes)

Offset	Typ	Bezeichnung	Beschreibung
000	Long	Next	Zeiger auf 2. 'RasInfo'-Struktur (nur wenn DualPlayfield benutzt wird)
004	Long	Bitmap	Zeiger auf 'Bitmap'-Struktur
008	Word	RxOffset	X-Position in der Bitmap (Normal =0) - für größere Bitmaps, um z.B. Scrolling zu erzeugen
010	Word	RyOffset	Y-Position in der Bitmap (Normal =0) s.o.
012		END	Ende der 'RasInfo'-Struktur

Um die BitMap mit der RasInfo-Struktur zu verbinden, muß man in 'BitMap' (Offset 4) die Anfangsadresse der BitMap-Struktur eintragen. Danach kann man die RasInfo-Struktur mit dem View-Port verbinden.

z. B.
```
move.l #bitmap,rasinfo+4
move.l #rasinfo,viewport+36
```

1.7 Wie man Strukturen initialisiert!

Bisher wurden die wichtigsten Strukturen, mit denen das System arbeitet, beschrieben. Es folgt jetzt eine Beschreibung der System-Routinen, mit denen man diese Strukturen initialisieren bzw. manipulieren kann.

```
Routine   :  InitBitMap (BitMap,depth,width,heigth) (A0,D0,D1,D2)
Library   :  graphics.library
Offset    :  -390 = -$186
Parameter:  A0 = Anfangsadresse der angelegten `BitMap`-Struktur
             D0 = Anzahl Bitplanes (Tiefe)
             D1 = Breite der Bitmap
             D2 = Höhe der Bitmap
Rückgabe :  keine
Erklärung:  Diese Routine initialisiert die angelegte `Bit-
             Map`-Struktur mit den wichtigsten Werten. Die Pla-
             nePtrX-Adressen müssen dann nur noch in der BitMap-
             Struktur von Ihnen selbst eingetragen werden.
```

Routine : InitRastPort (RastPort) (A1)
Library : graphics.library
Offset : -198 = -$c6
Parameter: A1 = Anfangsadresse der `RastPort`-Struktur die man angelegt hat
Rückgabe : keine
Erklärung: Diese Routine initialisiert die angelegte `RastPort`-Struktur mit den wichtigsten Werten

---

Routine : InitView (View) (A1)
Library : graphics.library
Offset : -360 = -$168
Parameter: A1 = Anfangsadresse der angelegten `View`-Struktur
Rückgabe : keine
Erklärung: Diese Routine initialisiert die angelegte `View`-Struktur mit den wichtigsten Werten

---

Routine : InitVPort (ViewPort) (A0)
Library : graphics.library
Offset : -204 = -$cc
Parameter: A0 = Anfangsadresse der angelegten `ViewPort`-Struktur
Rückgabe : keine
Erklärung: Diese Routine initialisiert die angelegte `ViewPort`-Struktur mit den wichtigsten Werten

---

Routine : MakeVPort (View,ViewPort) (A0,A1)
Library : graphics.library
Offset : -216 = -$d8
Parameter: A0 = Anfangsadresse der `View`-Struktur
A1 = Anfangsadresse der `ViewPort`-Struktur
Rückgabe : keine
Erklärung: Diese Routine erzeugt die einzelnen Copperliststrukturen und setzt diese in die entsprechenden Zeiger in der `ViewPort`-Struktur ein. Vorher muß natürlich die View-Port-Struktur mit der View-Struktur verbunden werden.

Routine : MrgCop (View) (A1)
Library : graphics.library
Offset : -210 = -$d2
Parameter: A1 = Anfangsadresse der `View`-Struktur
Rückgabe : keine
Erklärung: Diese Routine erzeugt aus den einzelnen Copperliststrukturen die mit `MakeVPort ()` erzeugt wurden eine einzige Copperlist-Struktur. Diese wird dann in die View-Struktur eingehängt (LongFrame-Copperliststruktur)

Routine : LoadView (View) (A1)
Library : graphics.library
Offset : -222 = -$de
Parameter: A1 = Anfangsadresse der `View`-Struktur
Rückgabe : keine
Erklärung: Diese Routine startet die Copperliste, auf die die Long-Frame-Copperlist-Struktur zeigt, über die Hardware. Und erzeugt somit das fertige Bild.

Routine : FreeVPortCopLists (ViewPort) (A0)
Library : graphics.library
Offset : -540 = -$21c
Parameter: A0 = Anfangsadresse der `ViewPort`-Struktur
Rückgabe : keine
Erklärung: Diese Routine gibt den Speicher wieder frei, der für die einzelnen Copperlisten durch die Routine `MakeVPort ()` vom System belegt wurde.

Routine : FreeCprList (CprList) (A0)
Library : graphics.library
Offset : -564 = -$234
Parameter: A0 = Anfangsadresse der LongFrame-Copperliststruktur, diese ist in der View-Struktur bei Offset 4 zu finden. Bei Hires muß auch ShortFrame-Copperliststruktur wieder freigegeben werden. (Offset 8)
Rückgabe : keine
Erklärung: Diese Routine gibt den Speicher wieder frei, der für die LongFrame-Copperliststruktur durch die Routine `MrgCop ()` vom System belegt wurde.

Routine : ViewAdress ()
Library : intuition.library
Offset : -294 = -$126
Parameter: keine
Rückgabe : in D0 = Anfangsadresse der aktuellen `View`-Struktur
Erklärung: Diese Routine gibt in D0 einen Zeiger auf die aktive `View`-Struktur, die vom System verwaltet wird, zurück.

Routine : ViewPortAdress (Window) (A0)
Library : intuition.library
Offset : -300 = -$12c
Parameter: A0 = Zeiger auf `Window`-Struktur
Rückgabe : in D0 = Anfangsadresse der aktuellen `ViewPort`-Struktur
Erklärung: Diese Routine gibt in D0 einen Zeiger auf die zu diesem Window gehörende `ViewPort`-Struktur zurück.

Routine : MakeScreen (Screen) (A0)
Library : intuition.library
Offset : -378 = -$17a
Parameter: A0 = Zeiger auf `Screen`-Struktur
Rückgabe : keine
Erklärung: Diese Routine führt die Funktion `MakeVPort ()` für den ViewPort des angegebenen Screen durch.

Routine : RethinkDisplay ()
Library : intuition.library
Offset : -390 = -$186
Parameter: keine
Rückgabe : keine
Erklärung: Ruft für alle Screens `MrgCop ()` und `LoadView ()` auf

Routine : RemakeDisplay ()
Library : intuition.library
Offset : -384 = -$180
Parameter: keine
Rückgabe : keine
Erklärung: Die Copperlisten werden für alle Screens neu berechnet [führt `MakeScreen ()` und danach `RethinkDisplay ()` aus]

Die Initialisierung einer RastPort-Struktur könnte z. B. so aussehen:

```
move.l gfxbase.a6        ; Basisadresse der graphics.library
move.l #rastport.a1      ; Zeiger auf RastPort-Struktur
jsr -198(a6)             ; Routine InitRastPort aufrufen
```

1.8 Das erste eigene Display

Da wir jetzt die wichtigsten Srukturen und Routinen des Systems kennen, werden wir mit deren Hilfe ein eigenes Display programmieren.

Um nach Beendigung des Programms, wieder auf das alte Display zurückschalten zu können, muß man als erstes den Zeiger (Anfangsadresse) auf die aktive View-Struktur speichern. Dazu rufen wir einfach die Routine 'ViewAdress ()' auf und speichern den Rückgabewert aus D0 in einem Puffer zwischen. Wenn man dann aufs alte Display zurück schalten will, ruft man lediglich die Routine 'LoadView ()' auf, der wir in A1 die Adresse des zuvor gespeicherten ViewPorts übergeben.

Bevor wir jedoch mit dem eigentlichen Programm beginnen können, müssen natrülich erst einmal alle Strukturen angelegt und soweit es geht mit Parametern versorgt werden. Danach öffnen wir die Grafik- und Intuition- Bibliothek und reservieren mit der Routine 'AllocMem ()' Speicher für die BitMaps.

Jetzt erst speichern wir den Zeiger des aktiven Views und initialisieren die angelegten Routinen.

Mit der Routine 'GetColorMap ()' lassen wir uns vom System eine ColorMap anlegen und speichern den Rückgabewert aus D0 zwischen.

Nachdem wir schon die Strukturen initialisiert haben, müssen wir nachträglich die Parameter Breite, Höhe und Modus in der ViewPort-Struktur setzen.

Nun können wir die Strukturen miteinander verbinden und durch aufrufen der drei Routinen 'MakeVPort ()', 'MrgCop ()' und 'LoadView ()' die neue Copperliste starten, wodurch unser Display auf dem Bildschirm sichtbar wird.

Wenn man sein Programm beenden möchte, sollte man natürlich den belegten Speicher wieder freigeben und auf das alte Display wieder zurückschalten.

Hier nochmal in Kurzform, die Schritte, welche durchzuführen sind, um ein eigenes Display zu erzeugen:

01. View-, ViewPort-, RasInfo-, ColorMap-, BitMap-, RastPort-Strukturen anlegen
02. graphics und intuition.library öffnen
03. Speicher für Bitmaps reservieren
04. Zeiger auf aktive View-Struktur speichern ['ViewAdress ()']
05. InitView () aufrufen
06. InitVPort () aufrufen
07. InitBitMap () aufrufen und InitRastPort ()
08. GetColorMap () aufrufen und Rückgabewert speichern [ColorMap-Struktur]
09. ViewPort-Parameter init. (Breite, Höhe und Modus)
10. Strukturen miteinander Verbinden
11. MakeVPort () aufrufen
12. MrgCop () aufrufen

13. LoadView () aufrufen
14. ...
14. ... Hier steht das weitere Programm ...
14. ...
;..... Mit folgenden Schritten wird aufs alte Display zurückgeschaltet

Exit:

15. LoadView () aufrufen [mit zuvor gespeicherter View-Adresse]
16. FreeColorMap () aufrufen
17. FreeVPortCopLists () aufrufen
18. FreeCprList () aufrufen
19. Speicher für Bitmaps wieder freigeben
20. graphics und intuition.library wieder schließen

Damit das erlernte nicht alles Theorie bleibt, folgt ein Programm, was obige Schritte durchführt. Also ein Primitiv-Display erzeugt.

```
;------------------------ Programmname = Display --------------------

 breite = 320     ; Bitmapbreite (muß durch 8 teilbar sein)
 höhe = 256       ; Bitmaphöhe
 tiefe = 5        ; Anzahl Bitplanes

;---------- Librarys öffnen ----------

 bsr openlibrarys

;---------- Speicher für Bitmaps reservieren ----------

 move.l #((breite/8)*tiefe*höhe),d0
 move.l #$10002,d1
 move.l 4,a6
 jsr -198(a6)                     ; AllocMem
 move.l d0,displaybase
```

```
 tst.l d0
 beq exit

;---------- aktiven View speichern ----------

 move.l intbase,a6
 jsr -294(a6)                    ; ViewAdress
 move.l d0,oldview

;---------- View-Struktur init. ----------

 move.l gfxbase,a6
 move.l #view,a1
 jsr -360(a6)                    ; InitView

;---------- ViewPort-Struktur init. ----------

 move.l gfxbase,a6
 move.l #viewport,a0
 jsr -204(a6)                    ; InitVPort

;---------- Bitmap-Struktur init. ----------

 move.l gfxbase,a6
 move.l #bitmap,a0
 move.l #tiefe,d0
 move.l #breite,d1
 move.l #höhe,d2
 jsr -390(a6)                    ; InitBitMap

;---------- Bitmap Pointer errechnen (PlanePrtX)----------

 move.l displaybase,a0           ; Basis des Display
 move.l #((breite/8)*höhe),d0    ; Höhe einer Bitmap
 move.w #tiefe-1,d1
 move.l #bitmap,a1
```

```
 add.l #8,a1
maploop:
 move.l a0,(a1)+
 add.l d0,a0
 dbra d1,maploop

;---------- Rasterport init. ----------

 move.l #rastport,a1
 move.l gfxbase,a6
 jsr -198(a6)                    ; InitRastPort

;---------- ColorMap-Struktur vom System anlegen lassen ----------

 move.l gfxbase,a6
 move.l #32,d0
 jsr -570(a6)                    ; GetColorMap
 move.l d0,colormap
 tst.l d0
 beq exit

;---------- Viewport-Parameter init. (Modus = Normal)----------

 move.w #breite,viewport+24
 move.w #höhe,viewport+26

;---------- Strukturen miteinander verbinden ----------
;---------- Viewport in View einhängen ----------

 move.l #viewport,view

;---------- Farbtabelle in Viewport einhängen ----------

 move.l colormap,viewport+4
```

```
;---------- RasInfo-Struktur in ViewPort-Struktur einhängen -------

 move.l #rasinfo,viewport+36

;---------- Bitmap-Struktur in RasInfo-Struktur einhängen ----------

 move.l #bitmap,rasinfo+4

;---------- Bitmap-Struktur in RastPort-Struktur einhängen ----------

 move.l #bitmap,rastport+4

;---------- Copperlisten konstrukieren ----------

 move.l gfxbase,a6
 move.l #view,a0
 move.l #viewport,a1
 jsr -216(a6)                        ; MakeVport

;---------- Copperlisten zu einer Copperliste zusammenfassen --------

 move.l gfxbase,a6
 move.l #view,a1
 jsr -210(a6)                        ; MrgCop

 move.w #1,copper                    ; Copperstatus = 1 , es wurden
                                     ; neue Copperlisten erstellt

;---------- Neue Copperliste starten ----------

 move.l gfxbase,a6
 move.l #view,a1
 jsr -222(a6)                        ; LoadView
```

```
;---------- Einen Text printen, damit man was sieht ----------

 move.l #rastport,a0
 move.l #text1,a1                    ; Zeiger auf Textstruktur
 move.l #00,d0                       ; x-position
 move.l #50,d1                       ; y-position
 move.l intbase,a6
 jsr -216(a6)                        ; PrintIText

main:
 btst #10,$dff016                    ; Rechte Maustaste gedrückt?
 bne main

exit:
 move.l oldview,a1                   ; auf alte Copperlist umschalten
 cmp.l #0,a1                         ; wurde alter View gespeichert?
 beq exit1
 move.l gfxbase,a6
 jsr -222(a6)                        ; alten View starten (LoadView)

exit1:
 move.l colormap,a0
 cmp.l #0,a0                         ; wurde eine ColorMap angelegt?
 beq exit1a
 move.l gfxbase,a6
 jsr -576(a6)                        ; ColorMap wieder freigeben
                                     ; FreeColorMap

;------ Speicher der einzelnen Copperlisten wieder freigeben ------

exit1a:
 cmp.w #0,copper                     ; wurden neue Copperlisten er
                                     ; stellt, und somit
 beq exit1b                          ; Speicher vom System belegt?
 move.l #viewport,a0                 ; erzeugte Copperlisten wieder
                                     ; freigeben
```

```
 move.l gfxbase,a6
 jsr -540(a6)                        ; FreeVPortCopLists
 move.l view+4,a0
 move.l gfxbase,a6
 jsr -564(a6)                        ; FreeCprList

exit1b:
 move.l displaybase,a1               ; wurde Speicher für BitMaps
 cmp.l #0,a1                         ; reserviert ?
 beq exit2
 move.l #((breite/8)*tiefe*höhe),d0
 move.l 4,a6
 jsr -210(a6)                        ; belegten Speicher freigeben
                                     ; (FreeMem)
exit2:
 bsr closelibrarys                   ; Librarys schließen
 rts

;---------- Unterroutinen ----------
;---------- Librarys öffnen ----------

openlibrarys:
 move.l 4,a6
 move.l #gfxname,a1
 clr.l d0
 jsr -552(a6)
 move.l d0,gfxbase
 move.l 4,a6
 move.l #intname,a1
 clr.l d0
 jsr -552(a6)
 move.l d0,intbase
 rts
```

```
;---------- Librarys schließen ----------

closelibrarys:
 move.l 4,a6
 move.l gfxbase,a1
 jsr -414(a6)
 move.l 4,a6
 move.l intbase,a1
 jsr -414(a6)
 rts

;---------- Parameter ----------

 gfxname: dc.b "graphics.library",0
 even
 intname: dc.b "intuition.library",0
 even
 gfxbase: dc.l 0
 intbase: dc.l 0
 oldview: dc.l 0
 colormap: dc.l 0                  ; Zeiger auf ColorMap-Struktur
 view: blk.b 18,0                  ; View-Struktur
 viewport: blk.b 40,0              ; ViewPort-Struktur
 rasinfo: blk.b 12,0               ; RasInfo-Sturktur
 rastport: blk.b 100,0             ; RastPort-Struktur
 bitmap: blk.b 40,0                ; Bitmap-Sturktur
 copper: dc.w 0
 displaybase: dc.l 0
 text1:                            ; Text-Struktur
 dc.b 1,0,0,0
 dc.w 0,0
 dc.l 0,title,0
 title: dc.b "Primitiv-Display (Press Right-Mouse)",0
 even
;-------------------------- Programmende -------------------------
```

1.9 Grafikmodi:

Wie aus der ViewPort-Struktur schon ersichtlich, besitzt der Amiga verschiedene Grafikmodi. Wenn man seinen Amiga normal startet, also über die Workbench, befindet man sich im Hires-Modus. Deswegen auch die hohe Auflösung in waagerechter Aufteilung.

Wie man nun die verschiedenen Modis programmiert, soll im folgenden beschrieben werden. Welchen man dann letztendlich für seine Programme wählt, bleibt dann Ihnen überlassen. Denn jeder Modus hat seine Vor- und Nachteile.

1.9.1 Hold and Modify (HAM)

Der Hold and Modify-Modus ist wohl der Bekannteste und Interessanteste. Durch ihn ist es Möglich die gesamten 4096 Farben des Amiga aufeinmal darzustellen.

Allerdings mit einem kleinen Nachteil. Dieser Modus kann nur im niedrig auflösenden Betrieb aktiviert werden und benötigt 6 Bitplanes. Wozu eine 6 BitPlane werden Sie sich bestimmt Fragen? Mit Hilfe dieser Bitplane stellt der Amiga nähmlich seine 4096 Farben zusammen.

Normalerweise sind die Übergänge von einem Punkt auf den anderen fließend und wechseln nicht sprunghaft. Diese Technik wird auch im HAM-Modus benutzt. Das heißt, das die Farbe des vorangegangenen Punktes von dem nachfolgenen modifiziert wird, also nur einen anderen Rot-, Grün oder Blauwert erhält. Jedoch kann man jeweils nur eine Farbkomponente ändern und nicht alle drei. Ein Farbwert kann logischerweise also erst nach 3 Punkten vollständig geändert werden.

Es besteht jedoch noch die Möglichkeit, durch ändern der Farbregister 0 - 15, die Farbe vollständig zu wechseln. Die Farben die ihre Farbwerte aus diesen ersten 16 Farbregistern entnehmen, werden als Echtfarben bezeichnet.

Wie wird nun eine Farbe angewählt, die nicht in diesen 16 Farbregistern enthalten ist?

Die Bits (4 und 5) aus den Bitplanes 5 und 6 bestimmen die Verwendung der unteren Bits (0 - 3 = BitPlanes 1- 4). Sind diese Bits gleich Null, wird ein Farbwert aus den Farbregistern 0 - 15 entnommen. Je nachdem wie die Bits 4 und 5 nun gesetzt sind, entscheidet darüber, welche Farbkomponente geändert werden soll. Die Farbkomponente erhält dann den Wert, der in den Bits 0 - 3 enthalten ist. Die anderen Farbkomponenten werden vom linksdanebenliegenden Punkt übernommen.

Wert der Bits 5 und 4:	**Farbkomponente die geändert wird:**
00	Keine; Farbwert aus Register 0 - 15
01	Blauwert aus Bits 0 - 3
10	Rotwert aus Bits 0 - 3
11	Grünwert aus Bits 0 - 3

Dieser Modus eignet sich lediglich für Titelbilder oder Diashows, also Grafiken die nicht animiert werden. Denn wenn man z. B. eine Figur über ein HAM-Bild animiert, kommt es zu Farbstreifen, es sei denn, man verwendet Sprites.

Um diesen Modus zu aktivieren, setzt man in der ViewPort-Struktur das Bit 11 und läßt den PlanePtr6-Zeiger, aus der BitMap-Struktur, auf eine 6 BitMap zeigen.

1.9.2 Hires

Der Hires-Modus halbiert sozusagen die Pixelbreite. Damit stehen einem 640 Punkte pro Zeile zur Verfügung. Wodurch man natürlich eine bessere Auflösung erhält.

In diesem Modus stehen einem allerdings max. nur 4 Bitplanes zur Verfügung, kann also max. 16 Farben ansteuern. Ein solches Display verbraucht doppelt soviel Speicher, wie ein normales. Ist ja auch logisch, schließlich steuert man nun 640 statt 320 Punkte an.

Dieser Modus wird durch das setzen von Bit 15, in der ViewPort-Struktur, aktiviert.

Hier ein Beispielprogramm, welches diesen Modus aktiviert:

```
;------------------------ Programmname = Hires --------------------

 breite = 640            ; Bitmapbreite (muß durch 8 teilbar sein)
 höhe = 256              ; Bitmaphöhe
 tiefe = 4               ; Anzahl Bitplanes

;---------- Librarys öffnen ----------

 bsr openlibrarys

;---------- Speicher für Bitmaps reservieren ----------

 move.l #((breite/8)*tiefe*höhe),d0
 move.l #$10002,d1
 move.l 4,a6
 jsr -198(a6)                        ; AllocMem
 move.l d0,displaybase
```

```
 tst.l d0
 beq exit

;---------- aktiven View speichern ----------

 move.l intbase,a6
 jsr -294(a6)                         ; ViewAdress
 move.l d0,oldview

;---------- View-Struktur init. ----------

 move.l gfxbase,a6
 move.l #view,a1
 jsr -360(a6)                         ; InitView


;---------- ViewPort-Struktur init. ----------

 move.l gfxbase,a6
 move.l #viewport,a0
 jsr -204(a6)                         ; InitVPort

;---------- Bitmap-Struktur init. ----------

 move.l gfxbase,a6
 move.l #bitmap,a0
 move.l #tiefe,d0
 move.l #breite,d1
 move.l #höhe,d2
 jsr -390(a6)                         ; InitBitMap

;---------- Bitmap Pointer errechnen (PlanePrtX) ----------

 move.l displaybase,a0                ; Basis des Display
 move.l #((breite/8)*höhe),d0         ; Höhe einer Bitmap
 move.w #tiefe-1,d1
```

```
 move.l #bitmap,a1
 add.l #8,a1

maploop:
 move.l a0,(a1)+
 add.l d0,a0
 dbra d1,maploop

;---------- Rasterport init. ----------

 move.l #rastport,a1
 move.l gfxbase,a6
 jsr -198(a6)                        ; InitRastPort

;---------- ColorMap-Struktur vom System anlegen lassen ----------

 move.l gfxbase,a6
 move.l #32,d0
 jsr -570(a6)              ; GetColorMap
 move.l d0,colormap
 tst.l d0
 beq exit

;---------- Viewport-Parameter init. ----------

 move.w #breite,viewport+24          ; Breite
 move.w #höhe,viewport+26            ; Höhe
 move.w #$8000,viewport+32           ; Modus = Hires

;---------- Strukturen miteinander verbinden ----------
;---------- Viewport in View einhängen ----------

 move.l #viewport,view
```

```
;---------- Farbtabelle in Viewport einhängen -----------

 move.l colormap,viewport+4

;---------- RasInfo-Struktur in ViewPort-Struktur einhängen --------

 move.l #rasinfo,viewport+36

;---------- Bitmap-Struktur in RasInfo-Struktur einhängen ----------

 move.l #bitmap,rasinfo+4

;---------- Bitmap-Struktur in RastPort-Struktur einhängen ---------

 move.l #bitmap,rastport+4

;---------- Copperlisten konstruieren ----------

 move.l gfxbase,a6
 move.l #view,a0
 move.l #viewport,a1
 jsr -216(a6)                       ; MakeVport

;---------- Copperlisten zu einer Copperliste zusammenfassen -------

 move.l gfxbase,a6
 move.l #view,a1
 jsr -210(a6)                       ; MrgCop
 move.w #1,copper                   ; Copperstatus = 1 , es wurden
                                    ; neue Copperlisten erstellt

;---------- Neue Copperliste starten ----------

 move.l gfxbase,a6
 move.l #view,a1
 jsr -222(a6)                       ; LoadView
```

```
;---------- Einen Text printen, damit man was sieht ----------

 move.l #rastport,a0
 move.l #text1,a1                   ; Zeiger auf Textstruktur
 move.l #00,d0                      ; x-position
 move.l #50,d1                      ; y-position
 move.l intbase,a6
 jsr -216(a6)                       ; PrintIText

main:
 btst #10,$dff016                   ; Rechte Maustaste gedrückt?
 bne main

exit:
 move.l oldview,a1                  ; auf alte Copperlist umschalten
 cmp.l #0,a1                        ; wurde alter View gespeichert?
 beq exit1
 move.l gfxbase,a6
 jsr -222(a6)                       ; alten View starten (LoadView)

exit1:
 move.l colormap,a0
 cmp.l #0,a0                        ; wurde eine ColorMap angelegt?
 beq exit1a
 move.l gfxbase,a6
 jsr -576(a6)                       ; ColorMap wieder freigeben
                                    ; FreeColorMap

;---------- Speicher der einzelnen Copperlisten wieder freigeben ---

exit1a:
 cmp.w #0,copper                    ; wurden neue Copperlisten er
                                    ; stellt, und  somit
 beq exit1b                         ; Speicher vom System belegt?
 move.l #viewport,a0                ; erzeugte Copperlisten wieder
                                    ; freigeben
```

```
 move.l gfxbase,a6
 jsr -540(a6)                       ; FreeVPortCopLists
 move.l view+4,a0
 move.l gfxbase,a6
 jsr -564(a6)                       ; FreeCprList

exit1b:
 move.l displaybase,a1              ; wurde Speicher für BitMaps
 cmp.l #0,a1                        ; reserviert ?
 beq exit2
 move.l #((breite/8)*tiefe*höhe),d0
 move.l 4,a6
 jsr -210(a6)                       ; belegten Speicher freigeben
                                    ; (FreeMem)
exit2:
 bsr closelibrarys                  ; Librarys schließen
 rts

;---------- Unterroutinen ----------
;---------- Librarys öffnen ----------

openlibrarys:
 move.l 4,a6
 move.l #gfxname,a1
 clr.l d0
 jsr -552(a6)
 move.l d0,gfxbase
 move.l 4,a6
 move.l #intname,a1
 clr.l d0
 jsr -552(a6)
 move.l d0,intbase
 rts
```

```
;---------- Librarys schließen ----------

closelibrarys:
 move.l 4,a6
 move.l gfxbase,a1
 jsr -414(a6)
 move.l 4,a6
 move.l intbase,a1
 jsr -414(a6)
 rts

;---------- Parameter ----------

gfxname: dc.b "graphics.library",0
 even
intname: dc.b "intuition.library",0
 even
gfxbase: dc.l 0
intbase: dc.l 0
oldview: dc.l 0
colormap: dc.l 0             ; Zeiger auf ColorMap-Struktur
view: blk.b 18,0             ; View-Struktur
viewport: blk.b 40,0         ; ViewPort-Struktur
rasinfo: blk.b 12,0          ; RasInfo-Sturktur
rastport: blk.b 100,0        ; RastPort-Struktur
bitmap: blk.b 40,0           ; Bitmap-Sturktur
copper: dc.w 0
displaybase: dc.l 0
text1:                       ; Text-Struktur
 dc.b 1,0,0,0
 dc.w 0,0
 dc.l 0,title,0
title: dc.b "Hires-Primitiv-Display ------  (Press Right-Mouse)",0
 even
;------------------------- Programmende -------------------------
```

1.9.3 Dualplayfield

In diesem Modus werden praktisch zwei Displays erzeugt, welche übereinander gelegt und unabhängig voneinander bewegt werden können.

Dieser Effekt kann z. B. für die Programmierung von Flugsimulatoren benutzt werden. Das eine Display zeigt die Instrumente usw. an, ist also unbewegt, im anderen kommt die Grafik auf einem zugescrollt. Man besitzt somit ein Fenster beliebiger Größe, durch das man hindurchschen kann.

Jedes der beiden Displays (Playfields) kann max. 3 Bitmaps besitzen. Ist eine ungerade Zahl von BitPlanes in Betrieb, besitzt das erste Display eine BitMap mehr. Es stehen einem somit auch nur max. 8 Farben pro Display zur Verfügung. Dabei bekommt das erste Display die Farbregister 0 - 7 und das zweite Display die Farbregister 8 - 15 zugeteilt.

Überall dort wo die Bitkombination gleich Null ist, wird nicht etwa die Farbe aus dem Hintergrundfarbregister 0 entnommen, sondern wird ein Punkt transparent dargestellt.

Um ein Dualplayfield zu erzeugen, setzt man das Bit 10 im 'Modus-Flag', in der ViewPort-Struktur und läßt den Zeiger 'Next' in der RasInfo-Struktur auf eine weitere RasInfo-Struktur zeigen. Denn in der zweiten RasInfo-Struktur ist ja wieder ein Zeiger auf eine BitMap-Struktur, welche einfach auf das zweite Display zeigt. Dieser Modus funktioniert auch im Interlace-Betrieb, nur stehen einem dann nur noch max. 4 Farben pro Display zur Verfügung.

Um die Dualplayfields getrennt voneinander zubewegen, definiert man einfach übergroße BitMaps, ändert die PlanePtrX-Zeiger in der entsprechenden BitMap-Struktur und berechnet danach mit 'MakeVPort ()', 'MrgCop ()' und 'LoadView ()' die Copperliste neu.

Zum besseren Verständnis folgt ein Programm, welches zwei Displays, von jeweils einer BitMap, erzeugt und das Erste von beiden mit den Cursortasten hoch und runter gescrollt werden kann.

```
;------------------ Programmname = Dualplayfield ------------------

 breite = 320           ; Bitmapbreite (muß durch 8 teilbar sein)
 höhe = 400             ; Bitmaphöhe
 tiefe = 2              ; Anzahl Bitplanes
 stops = höhe-256       ; Anzahl Zeilen die gescrollt werden können

;---------- Librarys öffnen -----------

 bsr openlibrarys

;---------- Speicher für Bitmaps reservieren ----------

 move.l #((breite/8)*tiefe*höhe),d0
 move.l #$10002,d1
 move.l 4,a6
 jsr -198(a6)                   ; AllocMem
 move.l d0,displaybase
 tst.l d0
 beq exit
```

```
;---------- aktiven View speichern ----------

 move.l intbase,a6
 jsr -294(a6)                    ; ViewAdress
 move.l d0,oldview

;---------- View-Struktur init. ----------

 move.l gfxbase,a6
 move.l #view,a1
 jsr -360(a6)                    ; InitView

;---------- ViewPort-Struktur init. ----------

 move.l gfxbase,a6
 move.l #viewport,a0
 jsr -204(a6)                    ; InitVPort

;---------- Bitmap(1)-Struktur init. (mit einer BitMap) ------------

 move.l gfxbase,a6
 move.l #bitmap,a0
 move.l #1,d0
 move.l #breite,d1
 move.l #höhe,d2
 jsr -390(a6)                    ; InitBitMap

;---------- Bitmap(2)-Struktur init. (mit einer BitMap) ----------

 move.l gfxbase,a6
 move.l #bitmap2,a0
 move.l #1,d0
 move.l #breite,d1
 move.l #höhe,d2
 jsr -390(a6)                    ; InitBitMap
```

```
;---------- Bitmap Pointer errechnen (PlanePrtX) ----------

 move.l displaybase,a0              ; Basis des Display
 move.l #((breite/8)*höhe),d0       ; Höhe einer Bitmap
 move.l a0,bitmap+8                 ; Zeiger auf erste Bitmap spei
                                    ; chern
 add.l d0,a0                        ; eine BitMaphöhe hinzuaddieren
 move.l a0,bitmap2+8                ; Zeiger auf zweite Bitmap spei
                                    ; chern

;---------- Rasterport init. ----------

 move.l #rastport,a1
 move.l gfxbase,a6
 jsr -198(a6)                       ; InitRastPort

;---------- ColorMap-Struktur vom System anlegen lassen ----------

 move.l gfxbase,a6
 move.l #32,d0
 jsr -570(a6)                       ; GetColorMap
 move.l d0,colormap
 tst.l d0
 beq exit

;---------- Viewport-Parameter init. ----------

 move.w #breite,viewport+24         ; Breite
 move.w #höhe,viewport+26           ; Höhe
 move.w #$400,viewport+32           ; Modus = Dualplayfield

;---------- Strukturen miteinander verbinden ----------
;---------- Viewport in View einhängen ----------

 move.l #viewport,view
```

```
;---------- Farbtabelle in Viewport einhängen ----------

 move.l colormap,viewport+4

;---------- RasInfo(1)-Struktur in ViewPort-Struktur einhängen -----

 move.l #rasinfo,viewport+36


;---------- Zweite RasInfo-Struktur mit Erster verbinden ----------

 move.l #rasinfo2,rasinfo

;---------- Bitmap-Struktur in RasInfo-Strukturen einhängen -------

 move.l #bitmap,rasinfo+4
 move.l #bitmap2,rasinfo2+4

;---------- Bitmap-Struktur(erste) in RastPort-Struktur einhängen -

 move.l #bitmap,rastport+4
 bsr load_copper                ; Copperliste starten

;---------- Einen Text auf Bitmap2 printen ----------

 move.l #bitmap2,rastport+4     ; Bitmap wechseln auf zweite
 move.l #rastport,a0
 move.l #text1,a1               ; Zeiger auf Textstruktur
 move.l #00,d0                  ; x-position
 move.l #100,d1                 ; y-position
 move.l intbase,a6
 jsr -216(a6)                   ; PrintIText
```

```
;---------- Einen Text auf Bitmap1 printen ----------

 move.l #bitmap,rastport+4          ; Bitmap wechseln auf erste
 move.l #rastport,a0
 move.l #text2,a1                   ; Zeiger auf Textstruktur
 move.l #00,d0                      ; x-position
 move.l #200,d1                     ; y-position
 move.l intbase,a6
 jsr -216(a6)                       ; PrintIText

main:
 btst #10,$dff016                   ; Rechte Maustaste gedrückt?
 beq exit
 move.l #bitmap,a0                  ; Zeiger auf 1. BitMap-Struktur
 move.l 8(a0),a0                    ; Adresse der ersten BitMap
 move.b $bfec01,d0                  ; Tastenwert holen
 cmp.b #$67,d0                      ; Cursor UP
 bne key2
 cmp.w #0,scrollpos                 ; schon Scrollanfang?
 beq main
 sub.w #1,scrollpos                 ; scrollpos um eins erniedrigen
 sub.l #(breite/8),a0               ; eine Zeile von der BitMap ab-
                                    ; ziehen
 move.l a0,bitmap+8                 ; eine Zeile nach Oben scrollen
 bsr wait_blank                     ; damit Scrolling ruckfrei
 bsr load_copper                    ; Copperliste neuberechnen
 bra main

key2:
 cmp.b #$65,d0                      ; Cursor Down
 bne main
 cmp.w #stops,scrollpos             ; Schon Scrollende?
 beq main
 add.w #1,scrollpos                 ; Scrollpos um eins erhöhen
 add.l #(breite/8),a0               ; eine Zeile zur Bitmap addieren
 move.l a0,bitmap+8                 ; eine Zeile nach unten scrollen
```

```
 bsr wait_blank                     ; damit Scrolling ruckfrei
 bsr load_copper                    ; Copperliste neuberechnen
 bra main

exit:
 move.l oldview,a1                  ; auf alte Copperlist umschalten
 cmp.l #0,a1                        ; wurde alter View gespeichert?
 beq exit1
 move.l gfxbase,a6
 jsr -222(a6)                       ; alten View starten (LoadView)
exit1:
 move.l colormap,a0
 cmp.l #0,a0                        ; wurde eine ColorMap angelegt?
 beq exit1a
 move.l gfxbase,a6
 jsr -576(a6)                       ; ColorMap wieder freigeben
                                    ; FreeColorMap

;---------- Speicher der einzelnen Copperlisten wieder freigeben --

exit1a:
 cmp.w #0,copper                    ; wurden neue Copperlisten er-
                                    ; stellt, und somit
 beq exit1b                         ; Speicher vom System belegt?
 move.l #viewport,a0                ; erzeugte Copperlisten wieder
                                    ; freigeben
 move.l gfxbase,a6
 jsr -540(a6)                       ; FreeVPortCopLists
 move.l view+4,a0
 move.l gfxbase,a6
 jsr -564(a6)                       ; FreeCprList
```

```
exit1b:
 move.l displaybase,a1              ; wurde Speicher für BitMaps
 cmp.l #0,a1                        ; reserviert ?
 beq exit2
 move.l #((breite/8)*tiefe*höhe),d0
 move.l 4,a6
 jsr -210(a6)                       ; Belegten Speicher freigeben
                                    ; (FreeMem)
exit2:
 bsr closelibrarys                  ; Librarys schließen
 rts

;---------- Unterroutinen ----------
;---------- Librarys öffnen ----------

openlibrarys:
 move.l 4,a6
 move.l #gfxname,a1
 clr.l d0
 jsr -552(a6)
 move.l d0,gfxbase
 move.l 4,a6
 move.l #intname,a1
 clr.l d0
 jsr -552(a6)
 move.l d0,intbase
 rts

;---------- Librarys schließen ----------

closelibrarys:
 move.l 4,a6
 move.l gfxbase,a1
 jsr -414(a6)
 move.l 4,a6
 move.l intbase,a1
```

```
 jsr -414(a6)
 rts

;---------- Copperlisten konstruieren -----------

load_copper:
 move.l gfxbase,a6
 move.l #view,a0
 move.l #viewport,a1
 jsr -216(a6)                    ; MakeVport

;---------- Copperlisten zu einer Copperliste zusammenfassen -------

 move.l gfxbase,a6
 move.l #view,a1
 jsr -210(a6)                    ; MrgCop
 move.w #1,copper                ; Copperstatus = 1 , es wurden
                                 ; neue Copperlisten erstellt

;---------- Neue Copperliste starten ----------

 move.l gfxbase,a6
 move.l #view,a1
 jsr -222(a6)                    ; LoadView
 rts

;---------- Auf Vertikal-Blank warten; damit kein Flackern ---------

wait_blank:
 move.l gfxbase,a6
 jsr -270(a6)                    ; WaitTOF
 rts
```

```
;---------- Parameter ----------

scrollpos: dc.w 0                  ; Wieviel Zeilen schon gescrollt
                                   ; wurden
gfxname: dc.b "graphics.library",0
 even
intname: dc.b "intuition.library",0
 even
gfxbase: dc.l 0
intbase: dc.l 0
oldview: dc.l 0
colormap: dc.l 0                   ; Zeiger auf ColorMap-Struktur
view: blk.b 18,0                   ; View-Struktur
viewport: blk.b 40,0               ; ViewPort-Struktur
rasinfo: blk.b 12,0                ; RasInfo-Sturktur
rasinfo2: blk.b 12,0
rastport: blk.b 100,0              ; RastPort-Struktur
bitmap: blk.b 40,0                 ; Bitmap-Sturktur
bitmap2: blk.b 40,0
copper: dc.w 0
displaybase: dc.l 0
text1:                             ; Text-Struktur
 dc.b 1,0,0,0
 dc.w 0,0
 dc.l 0,title,0
title: dc.b "DualPlayfield-Display 1 (Press)",0
 even
text2:                             ; Text-Struktur
 dc.b 9,0,0,0
 dc.w 0,0
 dc.l 0,title2,0
title2: dc.b "DualPlayfield-Display 2 (Mouse)",0
 even
;------------------------- Programmende -------------------------
```

1.9.4 Extra-Halfbrite

Dieser Modus funktioniert ähnlich wie der HAM-Modus. Auch hier werden 6 Bitplanes benötigt, also ist dieser Modus nur im niedrigauflösendem Betrieb möglich.

Mit 6 Bitplanes erhält man einen Wertebereich von 0 - 63. Der Amiga besitzt aber nur 32 Farbregister. Woher werden nun die anderen 32 Farben entnommen?

Dazu greift man einfach zu einem Trick:

Die ersten 5 Bits (0 - 4) werden weiterhin als Zeiger auf eines der 32 Farbregister benutzt. Wenn jetzt aber noch das 5. Bit aus der 6. Bitplane auf eins steht, dann wird der Farbwert aus den ersten 5 Bits, vor der Ausgabe auf dem Bildschirm, halbiert. Dies hat zur Folge, daß die Farbe in der halben Helligkeit erscheint. Damit stehen einem 64 Farben zur Verfügung.

Durch setzen von Bit 7 in 'Modes' in der ViewPort-Struktur und initialisieren von 6 Bitmaps in der Bitmap-Struktur, aktiviert man diesen Modus.

1.9.5 Scrolling

Um Scrolling zu erzeugen, definiert man als erstes eine übergroße Bitmap. Danach ändert man einfach, je nachdem in welche Richtung gescrollt werden soll, den RxOffset bzw. RyOffset in der RasInfo-Struktur und berechnet danach die Copperlisten neu.

Wenn man die Copperliste zu einer beliebigen Rasterstrahlpositi-on ändern würde, hätte dies ein unschönes Flackern zur Folge. Dies kann man verhindern, indem man, vor dem neuberechnen der Copperlisten, die Routine 'WaitTOF ()' oder 'WaitBOVP ()' oder 'VBeamPos ()' aufruft. Diese Routinen warten auf den Anfang bzw. das Ende einer Rasterstrahlzeile.

Scrolling kann man in jedem Modus erzeugen. Wird meist für die Spielcprogrammicrung angcwandt.

Das folgende Programm definiert zwei übergroße BitMaps, die mit den Cursortasten in alle 4 Richtungen gescrollt werden können.

```
;---------------------- Programmname = Scrolling --------------------

 breite = 400               ; Bitmapbreite (muß durch 8 teilbar sein)
 höhe = 400                 ; Bitmaphöhe
 tiefe = 2                  ; Anzahl Bitplanes

;---------- Librarys öffnen ----------

 bsr openlibrarys

;---------- Speicher für Bitmaps reservieren ----------

 move.l #((breite/8)*tiefe*höhe),d0
 move.l #$10002,d1
 move.l 4,a6
 jsr -198(a6)                     ; AllocMem
 move.l d0,displaybase
 tst.l d0
 beq exit
```

```
;---------- aktiven View speichern ----------

 move.l intbase,a6
 jsr -294(a6)                      ; ViewAdress
 move.l d0,oldview

;---------- View-Struktur init. ----------

 move.l gfxbase,a6
 move.l #view,a1
 jsr -360(a6)                      ; InitView

;---------- ViewPort-Struktur init. ----------

 move.l gfxbase,a6
 move.l #viewport,a0
 jsr -204(a6)                      ; InitVPort

;---------- Bitmap-Struktur init. ----------

 move.l gfxbase,a6
 move.l #bitmap,a0
 move.l #tiefe,d0
 move.l #breite,d1
 move.l #höhe,d2
 jsr -390(a6)                      ; InitBitMap

;---------- Bitmap Pointer errechnen (PlanePrtX) ----------

 move.l displaybase,a0             ; Basis des Display
 move.l #((breite/8)*höhe),d0      ; Höhe einer Bitmap
 move.w #tiefe-1,d1
 move.l #bitmap,a1
 add.l #8,a1
```

```
maploop:
 move.l a0,(a1)+
 add.l d0,a0
 dbra d1,maploop

;---------- Rasterport init. ----------

 move.l #rastport,a1
 move.l gfxbase,a6
 jsr -198(a6)                    ; InitRastPort

;---------- ColorMap-Struktur vom System anlegen lassen ----------

 move.l gfxbase,a6
 move.l #32,d0
 jsr -570(a6)                    ; GetColorMap
 move.l d0,colormap
 tst.l d0
 beq exit

;---------- Viewport-Parameter init. (Modus = Normal) ----------

 move.w #breite,viewport+24
 move.w #höhe,viewport+26

;---------- Strukturen miteinander verbinden -----------
;---------- Viewport in View einhängen ----------

 move.l #viewport,view

;---------- Farbtabelle in Viewport einhängen ----------

 move.l colormap,viewport+4
```

```
;---------- RasInfo-Struktur in ViewPort-Struktur einhängen ---------

 move.l #rasinfo,viewport+36

;---------- Bitmap-Struktur in RasInfo-Struktur einhängen -----------

 move.l #bitmap,rasinfo+4

;---------- Bitmap-Struktur in RastPort-Struktur einhängen ----------

 move.l #bitmap,rastport+4
 bsr new_copper

;---------- Einen Text printen, damit man was sieht ----------

 move.l #rastport,a0
 move.l #text1,a1                  ; Zeiger auf Textstruktur
 move.l #10,d0                     ; x-position
 move.l #150,d1                    ; y-position
 move.l intbase,a6
 jsr -216(a6)                      ; PrintIText
main:
 btst #10,$dff016                  ; Rechte Maustaste gedrückt?
 beq exit
 move.w rasinfo+8,d1               ; x-position nach d1
 move.w rasinfo+10,d2              ; y-position nach d2
 move.b $bfec01,d0                 ; Tastenwert holen
 cmp.b #$67,d0                     ; Cursorup ?
 bne key2
 cmp.w #0,d2                       ; y-pos am Anfang ?
 beq main
 sub.w #1,d2                       ; eine Zeile abziehen
 move.w d2,rasinfo+10              ; eine Zeile hoch scrollen
 bsr wait_tof                      ; Flackern ausschalten
 bsr new_copper                    ; Copperlisten neuberechnen
 bra main
```

```
key2:
 cmp.b #$65,d0                   ; Cursordown ?
 bne key3
 cmp.w #höhe-264,d2              ; y-pos am Ende ? (BitMaphö-
                                 ; he-(256+Fonthöhe))
 beq main
 add.w #1,d2                     ; eine Zeile dazuaddieren
 move.w d2,rasinfo+10            ; eine Zeile runter scrollen
 bsr wait_tof                    ; Flackern ausschalten
 bsr new_copper                  ; Copperlisten neuberechnen
 bra main

key3:
 cmp.b #$61,d0                   ; Cursorleft ?
 bne key4
 cmp.w #0,d1                     ; x-pos am Anfang ?
 beq main
 sub.w #1,d1                     ; einen Punkt abziehen
 move.w d1,rasinfo+8             ; einen Punkt nach links scrollen
 bsr wait_tof                    ; Flackern ausschalten
 bsr new_copper                  ; Copperlisten neuberechnen
 bra main

key4:
 cmp.b #$63,d0                   ; Cursorright ?
 bne main
 cmp.w #breite-320,d1            ; x-pos am Ende ?
 beq main
 add.w #1,d1                     ; einen Punkt dazuaddieren
 move.w d1,rasinfo+8             ; einen Punkt nach rechts scrol-
                                 ; len
 bsr wait_tof                    ; Flackern ausschalten
 bsr new_copper                  ; Copperlisten neuberechnen
 bra main
```

```
exit:
 move.l oldview,a1                  ; auf alte Copperlist umschalten
 cmp.l #0,a1                        ; wurde alter View gespeichert?
 beq exit1
 move.l gfxbase,a6
 jsr -222(a6)                       ; alten View starten (LoadView)

exit1:
 move.l colormap,a0
 cmp.l #0,a0                        ; wurde eine ColorMap angelegt?
 beq exit1a
 move.l gfxbase,a6
 jsr -576(a6)                       ; ColorMap wieder freigeben

;---------- Speicher der einzelnen Copperlisten wieder freigeben --

exit1a:
 cmp.w #0,copper                    ; wurden neue Copperlisten er-
                                    ; stellt, und somit
 beq exit1b                         ; Speicher vom System belegt?
 move.l #viewport,a0                ; erzeugte Copperlisten wieder
                                    ; freigeben
 move.l gfxbase,a6
 jsr -540(a6)                       ; FreeVPortCopLists
 move.l view+4,a0
 move.l gfxbase,a6
 jsr -564(a6)                       ; FreeCprList

exit1b:
 move.l displaybase,a1              ; wurde Speicher für BitMaps
 cmp.l #0,a1                        ; reserviert ?
 beq exit2
 move.l #((breite/8)*tiefe*höhe),d0
 move.l 4,a6
 jsr -210(a6)                       ; belegten Speicher freigeben
                                    ; (FreeMem)
```

```
exit2:
 bsr closelibrarys                    ; Librarys schließen
 rts

;---------- Unterroutinen ----------
;---------- Librarys öffnen ----------

openlibrarys:
 move.l 4,a6
 move.l #gfxname,a1
 clr.l d0
 jsr -552(a6)
 move.l d0,gfxbase
 move.l 4,a6
 move.l #intname,a1
 clr.l d0
 jsr -552(a6)
 move.l d0,intbase
 rts

;---------- Librarys schließen ----------

closelibrarys:
 move.l 4,a6
 move.l gfxbase,a1
 jsr -414(a6)
 move.l 4,a6
 move.l intbase,a1
 jsr -414(a6)
 rts

;---------- Copperliste neuberechnen ----------

new_copper:
 move.l gfxbase,a6
 move.l #view,a0
```

```
 move.l #viewport,a1
 jsr -216(a6)                   ; MakeVport
 move.l gfxbase,a6
 move.l #view,a1
 jsr -210(a6)                   ; MrgCop
 move.w #1,copper               ; Copperstatus = 1 , es wurden
                                ; neue Copperlisten erstellt
 move.l gfxbase,a6
 move.l #view,a1
 jsr -222(a6)                   ; LoadView
 rts


;---------- Flackern beim Scrollen ausschalten ----------

wait_tof:
 move.l gfxbase,a6
 jsr -270(a6)          ; WaitTOF
 rts

;---------- Parameter ----------

gfxname: dc.b "graphics.library",0
 even
intname: dc.b "intuition.library",0
 even
gfxbase: dc.l 0
intbase: dc.l 0
oldview: dc.l 0
colormap: dc.l 0                ; Zeiger auf ColorMap-Struktur
view: blk.b 18,0                ; View-Struktur
viewport: blk.b 40,0            ; ViewPort-Struktur
rasinfo: blk.b 12,0             ; RasInfo-Sturktur
rastport: blk.b 100,0           ; RastPort-Struktur
bitmap: blk.b 40,0              ; Bitmap-Sturktur
copper: dc.w 0
```

```
displaybase: dc.l 0
text1:                              ; Text-Struktur
 dc.b 1,0,0,0
 dc.w 0,0
 dc.l 0,title,0
title: dc.b "Scrolling-Demo (Press Cursor or Right-Mouse)",0
 even
;-------------------------- Programmende ------------------------
```

1.9.6 Double-Buffering - Blitzschnelle Grafik

Bei Double-Buffering werden zwei gleichgroße Displays benötigt. Während der Anwender gelangweilt in das sichtbare Display schaut, baut sich dic Grafik im unsichtbarcn Display auf. Danach aktiviert man das unsichtbare Display und die Grafik erscheint sofort.

Dieses Prinzip wird oft vcrwendet, um flackerfreie, blitzschnelle, rechenintensive Grafiken zu erzeugen. Hat allerdings den Nachteil, daß es sehr speicherraubend ist.

Programmierprinzip:

Man führt wie gewohnt die Schritte zur Erzeugung eines Displays durch. Nur diesmal werden zwei BitMap-Strukturen generiert, die jeweils auf ein Display zeigen. Natürlich müssen beide Displays gleich Groß sein. Jetzt erzeugt man wie gehabt mit den Routinen 'MakeVPort ()' und 'MrgCop ()' die Copperliste. Allerdings noch nicht 'LoadView ()' aufrufen. Denn nun wird erstmal der Zeiger auf die Copperliste, welcher bei Offset 4 (bei Interlace auch 8) in der View-Struktur zu finden ist, zwischengespeichert. Danach werden dann die Zeiger in der View-Struktur gelöscht, damit wieder eine neue erzeugt werden kann. Als nächstes wird jetzt die

zweite BitMap-Struktur für die erste in die RasInfo-Struktur eingehängt, danach wieder die Copperliste, wie oben beschrieben, erzeugt und zwischengespeichert.

Um nun von einem Display ins andere zu schalten, braucht nur die entsprechende Copperliste, welche vorher zwischengespeichert wurde, in die View-Struktur eingehängt und die Routine 'LoadView ()' aufgerufen werden.

Zum Besseren Verständnis das ganze nochmal in Kurzform:

1. Ein Display mit zwei BitMap-Strukturen erzeugen
2. Mit 'MakeVPort ()' und 'MrgCop ()' Copperliste für erste BitMap-Struktur generieren
3. Copperliste aus View-Struktur auslesen und speichern (Offset 4; bei Interlace auch 8)
4. Copperliste-Zeiger (Offset 4 und 8) in View-Struktur löschen (Wichtig)
5. zweite BitMap-Struktur in RasInfo-Struktur einhängen
6. erneut mit 'MakeVPort ()' und 'MrgCop ()' Copperliste für zweite BitMap-Struktur erzeugen
7. Copperliste aus View-Struktur auslesen (siehe Punkt 3)
8. Mit 'LoadView ()' jeweils entsprechende Copperliste starten

2. Kapitel

Copper - Der Coprozessor

- Copper - Der Coprozessor
- Die User-Copperliste
- Die Copperroutinen des Systems
- Programmierung des Coppers

2. Copper - Der Coprozessor

Wie schon im ersten Kapitel erwähnt wurde, gibt es zwei Möglichkeiten ein Bild darzustellen. Entweder setzt man die Bitmapadressen im Rasterinterrupt immer wieder auf den Anfangswert, damit auch gewährleistet ist, daß immer das gleiche Bild dargestellt wird, oder man läßt dieses den Copper erledigen.

Der Copper besitzt ein eigenes Programm, welches er abarbeitet. Dabei richtet er sich genau nach dem Rasterstrahlverlauf. Wenn der Rasterstrahl wieder beim Wert Null angekommen ist, wird automatisch das Programm erneut von vorne abgearbeitet.

Der Copper kennt drei Befehle. Den 'Move', 'Wait' und 'Skip'-Befehl. Wobei der 'Skip'-Befehl so uninteressant ist, das wir ihn außer Acht lassen können.

Der 'Move'-Befehl lädt ein Register eines Custom-Chips mit einem bestimmten Wert. Dabei stehen einem die Register von $DFF080 bis $DFF1BE zur Verfügung. Wenn das Register $DFF02E (COPCON) auf eins gesetzt wird, dann stehen einem auch noch die Register $DFF040 bis $DFF07E zur Verwendung. Leider ist es hier nicht möglich, jedes einzelne Register zu erläutern. Das würde den Rahmen dieses Buches sprengen. Eine ausführliche Beschreibung finden Sie z. B. im Intern-Buch.

Der 'Wait'- Befehl wartet solange, bis die angegebene Rasterzeile erreicht ist und fährt dann mit der Abarbeitung des Copperprogrammes fort.

Mit diesen beiden Befehlen ist man z. B. in der Lage sein Programm mit unwahrscheinlich vielen Specialeffekten zu versehen, wie meherere Hintergrundfarben oder die Darstellung von mehr als 8 Sprites gleichzeitig.

2.1 Die User-Copperliste

In der ViewPort-Struktur ist ein Zeiger auf eine UserCopperList-Struktur (UCopIns - Offset 20), der normalerweise auf Null ist. Wenn man diesen Zeiger auf eine Struktur zeigen läßt, die sich UCopList-Struktur nennt, danach die Copperliste mit den den gewünschten Befehlen füllt und neu berechnet, ist es möglich, den Copper zu programmieren. Diese Struktur, mit einer Länge von 12 Bytes, muß mit der Execroutine 'AllocMem ()' reserviert werden, weil das System, nach Beendigung des Programmes, den Speicher automatisch wieder zurückgibt. Ansonsten würde der GURU meditieren....

Beschreibung der Datenstruktur UCopList:

UCopList-Struktur (Länge = 12 Bytes - Mit 'AllocMem ()' reservieren):

Offset	Typ	Bezeichnung	Beschreibung
000	Long	NextUCopList	Zeiger auf nächste UserCop perlist-Struktur
004	Long		Zeiger auf erste Copperliste die von 'MakeVPort ()' angelegt wurde
008	Long		Zeiger auf dieser Struktur da zugehörigen Copperliste
012		END	Ende der 'UCopList'-Struktur

2.2 Die Copperroutinen des Systems

Wir kennen jetzt zwar die Copperbefehle und die dazugehörige Datenstruktur, jedoch wie fügt man einen Copperbefehl zur User-Copperliste?

Dazu stellt das System drei Routinen bereit. Die 'CMove ()', welche einen 'Move'-Befehl, die 'CWait ()', welche einen 'Wait'-Befehl hinzufügt. Und noch die 'CBump ()'-Routine. Diese muß nach jedem 'CMove ()' oder 'CWait ()' aufgerufen werden. Dadurch wird der interne Zeiger der User-Copperliste auf den nächsten Befehl erhöht.

Außerdem muß am Ende einer jeden User-Copperliste, ein 'Move'-Befehl erfolgen, welcher auf eine unmögliche Rasterstrahlposition wartet, damit die Copperliste wieder von vorne gestartet werden kann.

Es folgt nun eine Beschreibung der Routinen 'CBump()', 'CMove()' und 'CWait()':

```
Funktion :  CBump (UCopList) (A1)
Library  :  graphics.library
Offset   :  -366 = -$16e
Parameter:  A1 = Zeiger auf `UCopList`-Struktur die angelegt wurde
Rückgabe :  keine
Erklärung:  Diese Routine erhöht den internen Zeiger der UserCopper-
            liste. Muß nach jedem `CWait ()` und `CMove ()` aufgeru-
            fen werden!
-----------------------------------------------------------------

Funktion :  CMove (UCopList,Registernummer,Wert) (A1,D0,D1)
Library  :  graphics.library
Offset   :  -372 = -$174
```

```
Parameter:  A1 =  Zeiger auf `UCopList`-Struktur die angelegt wurde
            D0 =  Registernr. (z. B. #$180 für Hintergrundfarbregi-
                  ster) die Basisadresse wird nicht dabei übergeben
                  (Basis = $Dff000)
            D1 =  Wert (z. B. #$00 für schwarze Farbe)
Rückgabe :  keine
Erklärung:  Diese Routine führt einen `Move`-Befehl in die angelegte
            UserCopperliste ein.
```

```
Funktion :  CWait (UCopList,YPosition,XPosition) (A1,D0,D1)
Library  :  graphics.library
Offset   :  -378 = -$17a
Parameter:  A1 =  Zeiger auf `UCopList`-Struktur die angelegt wurde
            D0 =  Y-Position des Rasterstrahls auf die gewartet wer-
                  den soll
            D1 =  X-Position des Rasterstrahls auf die gewartet wer-
                  den soll nur in vierer Schritten möglich
Rückgabe :  keine
Erklärung:  Diese Routine führt einen `Wait`-Befehl in die angelegte
            UserCopperliste ein.
```

2.3 Programmierung des Coppers

Als erstes muß dazu die UserCopperliste angelegt und mit der ViewPort-Struktur (Offset 20) verbunden werden. Danach kann man mit den Routinen 'CMove ()', 'CWait ()' und 'CBump ()' die gewünschten Copperbefehle einfügen. Zum Schluß wartet man noch auf eine unmögliche Rasterstrahlposition.

Jetzt kann man endlich mit 'MrgCop ()' und 'LoadView ()' die erzeugte UserCopperliste zur echten hinzufügen.

Zum schnelleren Nachlesen, das ganze nochmal in Kurzform:

1. 'UCopList'-Struktur anlegen (Mit AllocMem ()')
2. Mit den Routinen 'CWait ()', 'CMove ()' und 'CBump ()' Copperliste programmieren (Beschreibung der Routinen siehe oben) Nach jedem 'CWait ()' oder 'CMove ()' muß mit 'CBump ()' der interne Zeiger innerhalb der UserCopperliste erhöht werden, damit Platz ist, für den nächsten Copperbefehl.
3. 'CWait ()' mit den Positionen Y=10000, X=256 aufrufen. Dadurch wartet der Copper auf eine unmögliche Rasterstrahlposition und springt damit wieder auf den Anfang der Copperliste.
4. 'UCopList'-Struktur in 'ViewPort'-Struktur einhängen (Offset 20)
5. 'MrgCop ()' und 'LoadView ()' ausführen

Selbstverständlich fehlt auch hier nicht das entsprechende Beispielprogramm:

```
;---------------------- Programmname = Copper ----------------------
;----------------- erzeugt zwei Hintergrundfarben ------------------

 breite = 320            ; Bitmapbreite (muß durch 8 teilbar sein)
 höhe = 256              ; Bitmaphöhe
 tiefe = 3               ; Anzahl Bitplanes

;---------- Librarys öffnen ----------

 bsr openlibrarys
```

```
;---------- alten View speichern ----------

 move.l intbase,a6
 jsr -294(a6)                        ; ViewAdress
 move.l d0,oldview

;---------- View-Struktur init. ----------

 move.l gfxbase,a6
 move.l #view,a1
 jsr -360(a6)                        ; InitView

;---------- ViewPort-Struktur init. ----------

 move.l gfxbase,a6
 move.l #viewport,a0
 jsr -204(a6)                        ; InitVPort

;---------- Viewport-Parameter init. ----------

 move.w #breite,viewport+24
 move.w #höhe,viewport+26

;---------- Viewport in View einhängen ----------

 move.l #viewport,view

;---------- ColorMap-Struktur vom System anlegen lassen ----------

 move.l gfxbase,a6
 move.l #32,d0
 jsr -570(a6)                        ; GetColorMap
 move.l d0,colormap
 tst.l d0
 beq exit
```

```
;---------- RasInfo-Struktur in ViewPort-Struktur einhängen ------

 move.l #rasinfo,viewport+36

;---------- Farbtabelle in Viewport einhängen ----------

 move.l colormap,viewport+4

;---------- Bitmap-Struktur init. ----------

 move.l gfxbase,a6
 move.l #bitmap,a0
 move.l #tiefe,d0
 move.l #breite,d1
 move.l #höhe,d2
 jsr -390(a6)                   ; InitBitMap

;---------- Speicher für Bitmaps reservieren ----------

 move.l #((breite/8)*tiefe*höhe),d0
 move.l #$10002,d1
 move.l 4,a6
 jsr -198(a6)                   ; AllocMem
 move.l d0,displaybase
 tst.l d0
 beq exit

;---------- Bitmap Pointer errechnen ----------

 move.l displaybase,a0          ; Basis des Display
 move.l #((breite/8)*höhe),d0   ; Höhe einer Bitmap
 move.w #tiefe-1,d1
 move.l #bitmap,a1
 add.l #8,a1
```

```
maploop:
 move.l a0,(a1)+
 add.l d0,a0
 dbra d1,maploop

;---------- Bitmap-Struktur in RasInfo-Struktur einhängen ----------

 move.l #bitmap,rasinfo+4

;---------- Rasterport init. ----------

 move.l #rastport,a1
 move.l gfxbase,a6
 jsr -198(a6)                   ; InitRastPort

;---------- Bitmap-Struktur in RastPort-Struktur einhängen ----------

 move.l #bitmap,rastport+4

;---------- Copperlisten konstruieren ----------

 move.l gfxbase,a6
 move.l #view,a0
 move.l #viewport,a1
 jsr -216(a6)                   ; MakeVport

;---------- Speicher für UCopList-Struktur reservieren ----------
;---------- (nicht mit blk.b 12,0) ----------

 move.l 4,a6
 move.l #$10002,d1              ; gelöschtes Chip-Ram
 move.l #12,d0                  ; 12 Bytes
 jsr -198(a6)                   ; AllocMem
 move.l d0,ucoplist             ; Adresse speichern
 tst.l d0                       ; Fehler aufgetreten?
 beq exit
```

```
;---------- User-Copper-List erstellen ----------

 move.l #0,d0                        ; y-pos.
 move.l #0,d1                        ; x-pos.
 bsr cwait                           ; Warten
 move.l #$180,d0                     ; Registernr. (Hintergrundfarbe)
 move.l #0,d1                        ; wert
 bsr cmove                           ; Wert speichern
 move.l #100,d0                      ; y-pos.
 move.l #0,d1                        ; x-pos.
 bsr cwait                           ; Warten
 move.l #$180,d0                     ; Registernr.
 move.l #1000,d1                     ; wert
 bsr cmove                           ; Wert speichern
 move.l #10000,d0                    ; unmögliche Rasterstrahlpos.
 move.l #256,d1                      ; muß am Ende jeder UserCopper-
                                     ; liste
 bsr cwait
 move.l ucoplist,viewport+20         ; UCopList in Viewport einhängen

;---------- Copperlisten zu einer Copperliste zusammenfassen -------

 move.l gfxbase,a6
 move.l #view,a1
 jsr -210(a6)                        ; MrgCop
 move.w #1,copper                    ; Copperstatus = 1 , es wurden
                                     ; neue Copperlisten erstellt

;---------- Neue Copperliste starten ----------

 move.l gfxbase,a6
 move.l #view,a1
 jsr -222(a6)                        ; LoadView
```

```
;---------- Einen Text printen, damit man was sieht ----------

 move.l #rastport,a0
 move.l #text1,a1
 move.l #00,d0
 move.l #50,d1
 move.l intbase,a6
 jsr -216(a6)                         ; PrintIText

main:
 btst #6,$bfe001                      ; Linke Maustaste
 bne main

exit:
 move.l oldview,a1                    ; auf alte Copperlist umschalten
 cmp.l #0,a1
 beq exit1
 move.l gfxbase,a6
 jsr -222(a6)                         ; LoadView

exit1:
 move.l colormap,a0
 cmp.l #0,a0
 beq exit1a
 move.l gfxbase,a6
 jsr -576(a6)                         ; FreeColorMap

;---------- Speicher der einzelnen Copperlisten wieder freigeben --

exit1a:
 cmp.w #0,copper                      ; wurden neue Copperlisten er
                                      ; stellt, und somit
 beq exit1b                           ; Speicher vom System belegt?
 move.l #viewport,a0                  ; erzeugte Copperlisten wieder
                                      ; freigeben
 move.l gfxbase,a6
```

```
 jsr -540(a6)                         ; FreeVPortCopLists
 move.l view+4,a0
 move.l gfxbase,a6
 jsr -564(a6)                         ; FreeCprList

exit1b:
 move.l displaybase,a1
 cmp.l #0,a1
 beq exit2
 move.l #((breite/8)*tiefe*höhe),d0
 move.l 4,a6
 jsr -210(a6)                         ; FreeMem

exit2:
 bsr closelibrarys
 rts

;---------- Librarys öffnen ----------

openlibrarys:
 move.l 4,a6
 move.l #gfxname,a1
 clr.l d0
 jsr -552(a6)
 move.l d0,gfxbase
 move.l 4,a6
 move.l #intname,a1
 clr.l d0
 jsr -552(a6)
 move.l d0,intbase
 rts
```

```
;---------- Librarys schließen ----------

closelibrarys:
 move.l 4,a6
 move.l gfxbase,a1
 jsr -414(a6)
 move.l 4,a6
 move.l intbase,a1
 jsr -414(a6)
 rts

;---------- CWait-Routine (D0 = y-pos., D1 = x-pos.) ----------

cwait:
 move.l gfxbase,a6
 move.l ucoplist,a1
 jsr -378(a6)                    ; CWait
 move.l gfxbase,a6
 move.l ucoplist,a1
 jsr -366(a6)                    ; CBump
 rts

;---------- CMove-Routine (D0 = registernr., D1 = wert) ----------

cmove:
 move.l gfxbase,a6
 move.l ucoplist,a1
 jsr -372(a6)                    ; CWait
 move.l gfxbase,a6
 move.l ucoplist,a1
 jsr -366(a6)                    ; CBump
 rts

;---------- Parameter ----------
gfxname: dc.b "graphics.library",0
 even
```

```
intname: dc.b "intuition.library",0
 even
gfxbase: dc.l 0
intbase: dc.l 0
oldview: dc.l 0
colormap: dc.l 0                    ; Zeiger auf ColorMap-Struktur
view: blk.b 18,0                    ; View-Struktur
viewport: blk.b 40,0                ; ViewPort-Struktur
rasinfo: blk.b 12,0                 ; RasInfo-Sturktur
rastport: blk.b 100,0               ; RastPort-Struktur
bitmap: blk.b 40,0                  ; Bitmap-Sturktur
ucoplist: dc.l 0                    ; Zeiger auf UCopList-Struktur
copper: dc.w 0
displaybase: dc.l 0
text1:
 dc.b 1,0,0,0
 dc.w 0,0
 dc.l 0,title,0
title: dc.b "Copper Raster-Demo (Left-Mousebutton)",0
 even
;-------------------------- Programmende -------------------------
```

3. Kapitel

Programmierung unter Intuition

- Programmierung unter Intuition
- Screen öffnen
- Screendatenstruktur
- Fenster öffnen
- Fensterdatenstruktur

3. Programmierung unter Intuition

Neben der Möglichkeit der Primitives-Grafikprogrammierung, gibt es noch eine zweite - Grafikprogrammierung unter Intuition.

Intuition ist genau wie die Primitives eine Bibliothek, in der sich eine Ansammlung von Routinen befindet. Jedoch kann man mit den Intuitionsroutinen wesentlicher einfacher und komfortabeler programmieren. Man kann z. B. kinderleicht Menüs, Windows, Screens usw. programmieren.

Wir wollen hier allerdings nur kurz auf die Programmierung von Screens und Windows eingehen.

3.1 Screen öffnen

Außer der im ersten Kapitel beschriebenen Möglichkeit, gibt es noch eine weitere, zur Erzeugung eines eigenen Displays.

Mit der Routine 'OpenScreen ()' aus der intuition.library. Beim Aufruf dieser Routine werden automatisch alle Strukturen, die für ein Display erforderlich sind, angelegt (RastPort, BitMap, RasInfo etc.). Diese Art erspart einem unwahrscheinlich viel Schreibarbeit, Speicher und zusätzlich lassen sich noch Windows auf diesem Screen verwalten.

Der Routine muß lediglich die Anfangsadresse einer Argumentenliste übergeben werden, in der alle nötigen Parameter enthalten sind. Denn woher soll Intuition wissen, wie breit oder wie hoch der Screen sein soll?

Wie nun im einzelnen die Struktur aufgebaut ist, entnehmen Sie bitte folgender Tabelle:

'OSArgs-Struktur: (Länge = 32 Bytes)

Wird für die Routine 'OpenScreen ()' benötigt!

Offset	Typ	Bezeichnung	Beschreibung
000	Word	X_Pos	X-Position des Screens im ViewPort (Normal = 0)
002	Word	Y_Pos	Y-Position des Screens im ViewPort
004	Word	Width	Breite der Bitmap
006	Word	Heigt	Höhe der Bitmap
008	Word	depth	Anzahl der Bitplanes (tiefe)
010	Byte	detail_pen	Farbregisternummer für Textfarbe
011	Byte	block_pen	Farbregisternummer für Hintergrundfarbe
012	Word	View_Modes	siehe ViewPort-Struktur
014	Word	Screen_Typ	siehe unten
016	Long	Font	Zeiger auf neuen Zeichensatz (0 = Normal)
020	Long	title	Zeiger auf Titletext, der mit Null endet
024	Long	Gadget	Zeiger auf Gadget-Struktur (0 = keine)
028	Long	BitMap	Zeiger auf eigene Bitmap-Struktur (nur wenn in Screen_Typ = Bit7 gesetzt ist, sonst Wert Null)
032		END	Ende der OSArgs-Struktur

Um seinen Screen z. B. mit einem Titletext zu versehen, setzt man in 'Title' (Offset 20) die Anfangsadresse des gewünschten Textes. Der Text muß allerdings mit einem Null-Byte enden.

z. B.

```
text: dc.b "Der neue title!",0
 even
```

Man kann alle möglichen Grafikmodis einstellen. Es ist jedoch das unter Kapitel 1 erwähnte zu beachten. Den Grafikmodi setzt man in 'View_Modes'.

Welchen Screen-Typ man aktivieren möchte, setzt man mit dem entsprechenen Bit in 'Screen_Typ'. Mögliche Bitkombinationen sind in folgender Tabelle enthalten:

Screen_Typ-Flag:

Bit	Wert	Bezeichnung	Beschreibung
0	$001	WBenchScreen	dies ist der Workbench-Screen
0-3	$00f	CustomScreen	Screen mit allen Funktionen
4	$010	ShowTitle	Bit wird gesetzt, wenn man 'ShowTitle ()' aufruft
5	$020	Beeping	Bit wird gesetzt, wenn man 'DisplayBeep ()' aufruft
6	$040	CustomBitMap	muß man setzen, wenn an seine eigene 'BitMap'- Struktur be nutzen will
7	$080	ScreenBehind	wenn gesetzt, Screen wird hin ter allen anderen Screens geöffnet
8	$100	ScreenQuiet	es erscheinen keine Gadgets und kein Title mehr auf dem Screen

3.2 Screendatenstruktur

Wenn man die Argumentenliste mit den gewünschten Werten gefüllt hat, übergibt man sie der 'OpenScreen ()'-Routine, welche schließlich den Screen dann 'öffnet'.

Zum öffnen des Screens fehlt jetzt nur noch die Beschreibung der Routine 'OpenScreen ()'.

```
Routine   :  OpenScreen (OSArgs) (A0)
Library   :  intuition.library
Offset    :  -198 = -$c6
Parameter:  A0 = Zeiger auf `OSArgs-Struktur`
Rückgabe :  in D0 = Zeiger auf `Screen`-Struktur [siehe unten]
Erklärung:  Diese Routine öffnet einen Screen mit den in der
            `OSArgs-`Struktur beschriebenen Funktionen
```

Natürlich exestiert parallel zum Öffnen eines Screens auch eine Routine, welche diesen wieder schließt:

```
Routine   :  CloseScreen (Screen) (A0)
Library   :  intuition.library
Offset    :  -66 = -$42
Parameter:  A0 = Zeiger auf `Screen-Struktur`, welchen man in D0
            durch die Routine `OpenScreen ()` erhält
Rückgabe :  keine
Erklärung:  Diese Routine schließt einen Screen auf den der Zeiger in
            A0 zeigt. Diesen Zeiger erhält man mit `OpenScreen ()`.
```

Die 'OpenScreen ()'-Routine übergibt in D0 einen Zeiger auf die Screendatenstruktur. In dieser Struktur sind alle nötigen Strukturen und Parameter enthalten die zum Aufbau eines Screens erforderlich sind. Wie z. B. die RastPort-Struktur, welche sehr wichtig für die Darstellung von Text und Bobs ist.

Wo nun die einzelnen Strukturen genau zu finden sind, wird durch die kurze Beschreibung der Screendatenstruktur ersichtlich:

'Screen'-Struktur: (Länge = 342 Bytes)

Erhält man in D0 zurück, wenn man 'OpenScreen ()' aufruft!

Offset	Typ	Bezeichnung	Beschreibung
000	Long	NextScreen	Zeiger auf nächsten Screen
004	Long	FirstWindow	Zeiger auf erstes Window auf diesem Screen
008	Word	X_Pos	x-Position des Screens
010	Word	Y_Pos	y-Position des Screens
012	Word	Width	Breite der Bitmap
014	Word	Heigth	Höhe der Bitmap
016	Word	MouseY	Y-Koordinate des Mauspointers
018	Word	MouseX	X-Koordinate des Mauspointers
020	Word	ScreenTyp	siehe Screen_Typ
022	Long	Title	Zeiger auf Screen-Titletext
026	Long	STitle	Zeiger auf Standart Titletext
030	Byte		Kopfleistenhöhe
031	Byte		vertikale Grenze der Kopfleiste
032	Byte		horizontale Grenze der Kopfleiste
033	Byte		vertikale Grenze des Menüs
034	Byte		horizontale Grenze des Menüs
035	Byte		oberer Fensterrahmen
036	Byte		linker Fensterrahmen
037	Byte		rechter Fensterrahmen
038	Byte		unterer Fensterrahmen
039	Byte	Pad	unbenutzt
040	Long	Font	Zeiger auf Font-Struktur

044	----	ViewPort	ab hier liegt die ViewPort-Struktur vom Screen
084	----	RastPort	ab hier liegt die RastPort-Struktur vom Screen
184	----	BitMap	ab hier liegt die BitMap-Struktur vom Screen
224	----	LayerInfo	ab hier liegt die LayerInfo-Struktur vom Screen
326	Long	GadGet	Zeiger auf erste Screen-Gadget-Struktur
330	Byte	DetailPen	Farbregisternummer für die Schriftfarbe
331	Byte	BlockPen	Farbregisternummer für die Hintergrundfarbe
332	Word	BackUp	Backup-Register für 'DisplayBeep ()'-Routine
334	Long	Extern	Zeiger auf externe Daten
338	Long	User	Zeiger auf User Daten
342		END	Ende der 'Screen'-Struktur

Folgendes Beispiellisting zeigt, wie man z. B. die Adresse des RastPorts ermittelt:

```
move.l screen,a0        ; Adresse der Screendatenstruktur nach A0
add.l #84,a0            ; A0 = Adresse vom RastPort
```

3.3 Fenster öffnen

Wenn man unter Intuition programmiert, ist man sogar in der Lage, darauf Fenster zu verwalten. Dieses ist wirklich kinderleicht und bietet einem unwahrscheinliche Möglichkeiten.

Die Routine 'OpenWindow ()' öffnet uns das gewünschte Fenster und 'CloseWindow ()' schließt es wieder. Man ist in der Lage sein Fenster nach belieben aufzubauen. Wir wollen hier jedoch nur auf das einfache öffnen und schließen eines Fensters eingehen.

Wie beim öffnen eines Screens, benötigen wir auch für das Fenster eine Argummentenliste, die man der Routine 'OpenWindow ()' übergeben muß. Diese Argumentenliste ist wie folgt aufgebaut:

'OWArgs'-Struktur: (Länge = 48 Bytes)

Wird für die Routine 'OpenWindow ()' benötigt!

Offset	**Typ**	**Bezeichnung**	**Beschreibung**
000	Word	X_Pos	X-Position des Windows auf dem Screen
002	Word	Y_Pos	Y-Position des Windows auf dem Screen
004	Word	Width	Breite des Windows
006	Word	Heigth	Höhe des Windows
008	Byte	detail_pen	Farbregisternummer für Textfarbe
009	Byte	block_pen	Farbregisternummer für Hintergrundfarbe
010	Long	IDCMP-Flags	siehe unten
014	Long	Window_Typ	siehe unten

018	Long	Gadget	Zeiger auf Gadget-Struktur (0 = keine)
022	Long	CheckMark	Zeiger auf eine Graphikstruktur für eigene Symbole zum Abhaken von Menüpunkten
026	Long	title	Zeiger auf Windowtitletext, der mit Null endet
030	Long	Screen	Zeiger auf eine 'Screen'-Struktur
034	Long	BitMap	Zeiger auf eigene Bitmap-Struktur (0 = keine)
038	Word	MinWidth	Mindestbreite des Fensters
040	Word	MinHeigth	Mindesthöhe des Fensters
042	Word	MaxWidth	Maximalbreite des Fensters
044	Word	MaxHeigth	Maximalhöhe des Fensters
046	Word	Screen_Typ	muß der selbe wie beim Screen sein
048		END	Ende der OWArgs-Struktur

Wie sich schon aus der Struktur ersehen läßt, braucht man für das Öffnen eines Fensters einen Screen auf dem das Fenster verwaltet werden kann. Dazu muß man die Adresse der Screendatenstruktur, welche man in D0 durch die Routine 'OpenScreen ()' erhält, in diese Struktur übertragen (Offset 30 - Screen).

z. B.:
```
move.l d0,OWArgs+30      ; D0 = Screendatenstruktur
```

Mit den IDCMP-Flags kann man dem System mitteilen bei welchen Ereignissen das System dem Programm eine Mitteilung senden soll, welche dann mit der Exec-Routine 'GetMsg ()' abgefragt werden kann (z. B. beim Anklicken des Windowschließsymbols usw.).

Hier die möglichen IDCMP-Flags:

IDCMP-Flags: diese Bits bestimmen, bei welchen Ereignissen Intuition dem Programm eine Meldung übermitteln soll!

Bit	Wert	Bezeichnung	Nachricht bei:
00	$000001	SizeVerify	
01	$000002	NewSize	Veränderung der Fenstergröße
02	$000004	RefreshWindow	erneuern eines Windows
03	$000008	MouseButtons	drücken einer Maustaste
04	$000010	MouseMove	Bewegungen der Maus
05	$000020	GadGetDown	Auswahl eines speziellen Gad gets
06	$000040	GadgetUp	s.o. (aber wenn linke Maustaste losgelassen wird)
07	$000080	ReqSet	Erscheinung eines Requesters
08	$000100	MenuPick	Auswahl eines Menüpunktes
09	$000200	CloseWindow	schließen des Fensters
10	$000400	RawKey	drücken einer Taste
11	$000800	ReqVerify	
12	$001000	ReqClear	verschwinden eines Requesters
13	$002000	MenuVerify	
14	$004000	NewPrefs	ändern der Preferences
15	$008000	DiskInserted	einlegen einer Diskette
16	$010000	DiskRemoved	Herausnehmen einer Diskette
17	$020000	WBenchMessage	
18	$040000	ActiveWindow	aktivieren des Windows
19	$080000	InActiveWindow	desaktivieren des Windows
20	$100000	DeltaMove	Mausbewegungen relativ mel den
21	$200000	VanillaKey	
22	$400000	IntuiTicks	

Was für Eigenschaften soll nun unser Fenster aber besitzen?

Dieses können wir mit den Window-Typ-Flag bestimmen. Ob unser Fenster verschiebbar, vergrößerbar etc. sein soll.

Window_Typ:

die Bits bestimmen die Eigenschaften eines Fensters!

Bit	Wert	Bezeichnung	Beschreibung
00	$0000001	WindowSizing	Fenstergröße veränderbar
01	$0000002	WindowDrag	Fenster verschiebbar
02	$0000004	WindowDepth	Fensterüberlagerung möglich
03	$0000008	WindowClose	besitzt ein Fensterschließsymbol
04	$0000010	SizeBright	Vergrößerungsgadget ist rechts
05	$0000020	SizeBBottom	Vergrößerungsgadget ist unten
06	$0000040	SimpleRefresh	Neuzeichnen manuell
07	$0000080	SuperBitMap	ganzen Fensterinhalt speichern
08	$0000100	BackDrop	Fenster nach Hinten
09	$0000200	ReportMouse	Mauskoordinaten melden
10	$0000400	GimmeZeroZero	Fenster ohne Leiste
11	$0000800	BorderLess	Fenster ohne Ränder
12	$0001000	Activate	Fenster sofort aktiv
13	$0002000	WindowAktive	wird von Intuition gesetzt
14	$0004000	InRequest	wird von Intuition gesetzt
15	$0008000	MenuState	wird von Intuition gesetzt
16	$0010000	RMBTrap	bei drücken der rechten Maustaste kein Menü
17	$0020000	NoCareRefresh	keine Erneuerungsmeldung
24	$1000000	WindowRefresh	wird von Intuition gesetzt
25	$2000000	WBenchWindow	wird von Intuition gesetzt
26	$4000000	WindowTicked	wird von Intuition gesetzt

3.4 Fensterdatenstruktur

Jetzt fehlt uns zum Öffnen des Fensters nur noch die Beschreibung der Routinen 'OpenWindow ()' und 'CloseWindow ()':

```
Routine   :  OpenWindow (OWArgs) (A0)
Library   :  intuition.library
Offset    :  -204 = -$cc
Parameter:  A0 = Zeiger auf `OWArgs-Struktur`
Rückgabe :  in D0 = Zeiger auf `Window`-Struktur [siehe unten]
Erklärung:  Diese Routine öffnet ein Window auf dem in der OWArgs-
            Struktur angegebenem Screen, mit den gewünschten Funk-
            tionen
```

```
Routine   :  CloseWindow (Window) (A0)
Library   :  intuition.library
Offset    :  -72 = -$48
Parameter:  A0 =  Zeiger auf `Window-Struktur`, welchen man in D0
                  durch die Routine `OpenWindow ()` erhält
Rückgabe :  keine
Erklärung:  Diese Routine schließt ein Window auf den der Zeiger in
            A0 zeigt. Diesen Zeiger erhält man mit `OpenWindow ()`.
```

Durch die Routine 'OpenWindow ()' erhalten wir in D0 einen Zeiger auf, die zu dem gerade geöffnetem Window gehörende Windowdatenstruktur. In dieser Struktur sind alle nötigen Strukturen und Parameter enthalten, die zum verwalten eines Fensters nötig sind.

Wie diese Datenstruktur im einzelnen aufgebaut ist, soll hier auch nur in Kurzform erwähnt werden. Damit Sie auch wissen, wo wir die Offsets für bestimmte Strukturen, welche wir in einigen Programmen, die in späteren Kapiteln folgen, herbekommen.

'Window'-Struktur: (Länge = 124 Bytes)

Erhält man in D0 zurück, wenn man 'OpenWindow ()' aufruft!

Offset	Typ	Bezeichnung	Beschreibung
000	Long	NextWindow	Zeiger auf nächste Fenster struktur
004	Word	x-Pos	x-Position des Windows, relativ zum Screen
006	Word	y-Pos	y-Position des Windows, relativ zum Screen
008	Word	Width	Breite des Fensters
010	Word	Heigth	Höhe des Fensters
012	Word	DeltaMouseY	Y-Koordinate der Maus, relativ zum Fenster
014	Word	DeltaMouseX	X-Koordinate der Maus, relativ zum Fenster
016	Word	MinWidth	minimale Breite des Fensters
018	Word	MinHeigth	minimale Höhe des Fensters
020	Word	MaxWidth	maximale Breite des Fensters
022	Word	MaxHeigth	maximale Höhe des Fensters
024	Long	WindowTyp	siehe oben (Window_Typ)
028	Long	Menu	Zeiger auf Menüstruktur
032	Long	Title	Zeiger auf TitleText
036	Long	FirstRequest	Zeiger auf erste aktive Reque sterstruktur
040	Long	DKRequest	Zeiger auf Double-Click-Re questerstruktur
044	Word	BRequest	Anzahl der Requester, die das Fenster sperren
046	Long	Screen	Zeiger auf 'Screen'-Struktur
050	Long	RastPort	Zeiger auf 'RastPort'-Struktur des Fensters
054	Byte		Linker Rahmen
055	Byte		Oberer Rahmen

056	Byte		Rechter Rahmen
057	Byte		Unterer Rahmen
058	Long	RRastPort	Zeiger auf Rahmen-RastPort-Struktur
062	Long	Gadget	Zeiger auf erste Gadget-Struk tur
066	Long		Zeiger auf Eltern-Fenster-Struktur
070	Long		Zeiger auf Kind-Fenster-Struk tur
074	Long	SprData	Zeiger auf SpriteData für MausPointer
078	Byte	SprHeigth	Höhe des Spritepointers
079	Byte	SprWidth	Breite des Spritepointers (max. 16)
080	Byte	XOffset	x-Offset des Spritepointers
081	Byte	YOffset	y-Offset des Spritepointers
082	Long	IDCMP-Flag	siehe oben
086	Long	UMessagePort	User-Message-Port
090	Long	WMessagePort	Fenster-Message-Port
094	Long	MessageKey	Intuition-Message-Port
098	Byte	DetailPen	Farbregisternummer der Schriftfarbe
099	Byte	BlockPen	Farbregisternummer der Hinter grundfarbe
100	Long		Zeiger auf eigene Me nü-Hakensymbole-Struktur
104	Long	STitle	Zeiger auf Screen-Titletext
108	Word	GZZ-MausX	
110	Word	GZZ-MausY	
112	Word	GZZ-Width	
114	Word	GZZ-Heigth	
116	Long	Extern	Zeiger auf Externe Daten
120	Long	User	Zeiger auf User Daten
124		END	Ende der 'Window'-Struktur

Am Ende dieses Kapitels noch ein Beispielprogramm, welches einen Screen und ein Fenster darauf öffnet. Es wartet dann solange bis das Fensterschließsymbols aktiviert wird, so daß daraufhin das Fenster geschlossen wird. Mit der rechten Maustaste kann das Programm beendet werden.

```
;----------------------- Programmname = Screen --------------------

start:
 move.l #intname,a1              ; Zeiger auf Intuitionname
 clr.l d0                        ; Version = 0
 move.l 4,a6                     ; Execbase nach A6
 jsr -552(a6)                    ; OpenLibrary
 move.l d0,intbase               ; Intuitionbasis speichern
 tst.l d0                        ; Error ?
 beq exit

 move.l #OSArgs,a0               ; Zeiger auf Screenargumentenli
                                 ; ste
 move.l intbase,a6               ; Intuitionbasis nach A6
 jsr -198(a6)                    ; OpenScreen
 move.l d0,screen                ; Adresse der Screendatenstruktur
                                 ; speichern
 tst.l d0                        ; Error ?
 beq exit

 move.l #OWArgs,a0               ; Zeiger auf Windowargumentenli
                                 ; ste
 move.l intbase,a6               ; Intuitionbasis nach A6
 jsr -204(a6)                    ; OpenWindow
 move.l d0,window                ; Adresse der Windowdatenstruktur
                                 ; speichern
 tst.l d0                        ; Error ?
 beq exit
```

```
main:
 btst #10,$dff016                   ; rechte Maustaste gedrückt ?
 beq exit
 bsr message                        ; Ereignis abfragen
 bra main

;---------- Message Abfragen ----------

message:
 move.l window,a0                   ; Windowdatenstruktur
 cmp.l #0,a0                        ; liegt eine Adresse vor ?
 beq message_exit
 move.l 86(a0),a0                   ; Zeiger auf Window-UserPort
 move.l 4,a6                        ; Execbasis
 jsr -372(a6)                       ; GetMsg ()
 tst.l d0                           ; wurde Message gesendet?
 beq message_exit
 move.l d0,a0
 move.l 20(a0),d0                   ; D0 = enthält IDCMP-Wert
 cmp.l #$200,d0                     ; WindowClose ?
 bne message_exit
 bsr closewindow                    ; Window schließen

message_exit:
 rts

;---------- Window schließen ----------

closewindow:
 move.l window,a0                   ; Windowdatenstruktur
 cmp.l #0,a0                        ; liegt eine Adresse vor ?
 beq cw1
 move.l intbase,a6                  ; Intuitionbasis
 jsr -72(a6)                        ; CloseWindow
 move.l #0,window                   ; Window-Speicherstelle löschen
```

```
cw1:
 rts

;---------- Programm beenden ----------

exit:
 bsr closewindow                ; Window schließen
 move.l screen,a0               ; Screendatenstruktur
 cmp.l #0,a0                    ; liegt eine Adresse vor ?
 beq exit2
 move.l intbase,a6              ; Intuitionbasis
 jsr -66(a6)                    ; CloseScreen

exit2:
 move.l intbase,a1              ; Intuitionbasis
 cmp.l #0,a1                    ; liegt eine Adresse vor ?
 beq exit3
 move.l 4,a6                    ; Execbasis
 jsr -414(a6)                   ; CloseLibrary

exit3:
 rts                            ; Programmende


;---------- Parameter -----------

intname: dc.b "intuition.library",0
 even
intbase: dc.l 0
window: dc.l 0
OSArgs:                         ; ab hier Screenargumentenliste
 dc.w 0,0                       ; x und y-position
 dc.w 320,256                   ; Breite und Höhe
 dc.w 5                         ; Anzahl der Bitplanes (=32 Far
                                ; ben)
 dc.b 1,3                       ; Textfarben
```

```
 dc.w 0                                 ; ViewMode = Normal
 dc.w 15                                ; Screen mit allen Funktionen
 dc.l 0                                 ; kein eigener Font
 dc.l title                             ; Zeiger auf Titletext
 dc.l 0                                 ; keine Gadget
 dc.l 0                                 ; keine eigene Bitmap-Struktur
title: dc.b "Press Right Mouse to Exit",0
 even

OWArgs:                                 ; ab hier Windowargumentenliste
 dc.w 0,40                              ; x und y-position
 dc.w 130,50                            ; Breite und Höhe
 dc.b 1,2                               ; Textfarben
 dc.l $200                              ; IDCMP-Flags (CloseWindow)
 dc.l $100b                             ; Window-Typ-Flag
 dc.l 0                                 ; kein Gadget
 dc.l 0                                 ; keine CheckMark
 dc.l wtitle                            ; Zeiger auf Windowtitletext

screen:
 dc.l 0                                 ; hier wird Screendatenstruktur
                                        ; gespeichert
 dc.l 0                                 ; keine eigene Bitmapstruktur
 dc.w 50,40,150,100                     ; Begrenzungsgrößen
 dc.w 15                                ; Screentyp
wtitle: dc.b "Ein Window",0
 even
;------------------------- Programmende ---------------------------
```

4. Kapitel

Unterbrechungen - Interrupts

- Unterbrechungen - Interrupts
- User-Interrupt
- Raster-Interrupt

4. Unterbrechungen - Interrupts

Oft ist es notwendig, irgendwelche Zustände bestimmter Register zu überprüfen oder man möchte einfach nur bestimmte Routinen in regelmäßigen Zeitabständen anspringen, um z. B. ruckfreie Animation ablaufen zu lassen.

Man kann dazu natürlich in seinem Programm, die laufende Zeit abfragen und daraufhin eine bestimmte Routine anspringen. Aber selbst diese Möglichkeit wäre noch viel zu ungenau. Außerdem ist sie sehr rechenintensiv und kostet Speicher.

Hierfür exestieren die Interrupts. Interrupts - kurz IRQ - sind wie der Name schon sagt, Unterbrechungen. Sie unterbrechen das laufende Programm bei bestimmten Ereignissen, führen eine angegebene Routine aus und springen danach wieder ins unterbrochende Programm, welches wieder fortgesetzt wird.

Der Amiga bietet einem die Wahl zwischen 15 verschiedenen Ereignissen, bei denen ein IRQ ausgelöst wird. Es stehen einem jedoch nur 7 verschiedene Interruptebenen zur Verfügung. Eine Interruptebene ist nichts weiter als eine Adresse, in der die Anfangsadresse der Routine gespeichert ist, welche bei einer Unterbrechung ausgeführt werden soll.

Sie werden sich bestimmt fragen, welcher IRQ welche Ebene zugeteilt bekommt?

Dazu bekommt jeder der 15 möglichen IRQs eine Priorität zwischen 1 und 7 zugeteilt. Alle IRQs mit gleicher Priorität werden zu einer Ebene zusammengefaßt. Diese Priorität wird als CPU-Priorität bezeichnet.

Aber welcher IRQ einer gleichen Ebene wird nun zuerst ausgeführt?

Dafür erhält jeder IRQ noch eine Pseudo-Priorität zwischen 0 und 14. Der IRQ welcher eine hähere Pseudo-Priorität besitzt, wird dann zuerst abgearbeitet.

Woher soll nun der Amiga wissen, wo die Adresse der gewünschten Ebene zu finden ist?

Diese Adressen liegen beim Amiga immer an der selben Stelle, beginnend bei $64 bis $7c.

Das Ganze zur Verdeutlichung nochmal in Tabellenform:

Interrupt- Priorität

Name:	**CPU:**	**Pseudo:**	**Ebene:**	**Ereignis:**
NMI	7	15	$7C	(Nicht maskierter IRQ)
EXTER	6	13	$78	Signal vom CIA-B
DiskSync	5	12	$74	Sync-Word der Disk paßt
RBF	5	11	$74	Serieller Input
Audio3	4	10	$70	Audiokanal 3 Daten am Ende
Audio2	4	09	$70	Audiokanal 2 Daten am Ende
Audio1	4	08	$70	Audiokanal 1 Daten am Ende
Audio0	4	07	$70	Audiokanal 0 Daten am Ende
BLIT	3	06	$6c	Blitteroperation fertig
VertBlank	3	05	$6c	neues Bild fängt an
Copper	3	04	$6c	Copperprogramm am Ende angelangt
Ports	2	03	$68	Signal von CIA-A
SoftInt	1	02	$64	Software IRQ
DSKBLK	1	01	$64	Disk-DMA fertig
TBE	1	00	$64	Serielle Ausgabe

Es sind natürlich nicht immer alle IRQs angeschaltet, was vielzuviel rechenzeit kosten würde. Sondern immer nur ganz bestimmte, wie z. B. der DSKBLK-IRQ, welcher immer dann angesprungen wird, wenn man Daten von Diskette lädt. Lediglich der Vert-Blank-IRQ ist immer angeschaltet. Denn der sorgt dafür, das 50 mal in der Sekunde das selbe Bild aufgebaut wird.

Welche IRQs nun erlaubt werden dürfen und welche nicht, kann man durch setzen bestimmter Bits im INTENA-Register ($DFF09A) einstellen. Wenn ein IRQ gestartet wird, was durch setzen eben erwähnter Bits geschieht, wird im INTREQ-Register ($DFF09C) das dazugehörige Bit gesetzt. Dieses Bit muß man in seiner IRQ-Routine, deren Adresse man in der entsprechenden Ebene eingetragen hat, wieder zurücksetzen. Ansonsten würde nie wieder das eigentliche Programm fortgesetzt werden. Jede IRQ-Routine muß auch mit einem RTE enden, damit auch wieder zum eigentlichen Programm zurückgesprungen werden kann. Eine genaue Beschreibung der Register entfällt hier, da wir uns ja nur mit der Systemprogrammierung beschäftigen wollen und nicht mit der Hardware.

4.1 User-Interrupt

Ein User-Interrupt, welcher auch als Softinterrupt bezeichnet wird, ist schon automatisch vom System initialisiert.

Wir wissen ja, das die Routine welche bei einem Soft-Interrupt angesprungen wird, in der Ebene 1, also Adresse $64, zu finden ist. Wir brauchen also lediglich diese Adresse auszulesen und zwischenzuspeichern. Danach tragen wir dann die Adresse unserer eigenen IRQ-Routine dort ein. Am Ende unserer IRQ-Routine springen wir dann wieder zur eigentlichen Systemroutine, welche dann den IRQ mit RTE beendet.

Diese Methode ist eigenlich gar nicht erlaubt, denn sie ist nichts halbes und nichts ganzes. Wir programmieren nicht über die Hardware aber auch nicht über die Systemroutinen.

Diese Methode funktioniert jedoch einbahnfrei und ist zudem noch die kürzeste und einfachste. Denn wir erweitern einfach die System-IRQ-Routinen. Der Nachteil ist aber, das wir nicht jede IRQ-Ebene so ansteuern können, sondern nur die, die auch vom System schon initialisiert wurden. (z. B. Ebene 1 oder 3)

Hier nun das Ganze als gut dokumentiertes Listing:

```
;--------------------- Programmname = SoftIRQ ---------------------
;------------------- wartet auf rechte Maustaste -------------------
;------------------- läßt den Bildschirm blinken -------------------

start:
 move.l $64,oldirq+2       ; alte IRQ-Adresse auslesen und speichern
 move.l #softirq,$64       ; eigene IRQ-Adresse in Ebene 1 eintragen

wait:
 btst #10,$dff016                ; wurde rechte Maustaste gedrückt
 bne wait                        ; wenn nicht dann weiter abfragen

exit:
 move.l oldirq+2,$64             ; alte IRQ-Adresse wieder zu-
                                 ; rückschreiben
 rts                             ; Programm beenden

;--------- Hier beginnt IRQ-Routine ---------

softirq:
 movem.l d0-d7/a0-a6,-(sp)       ; Register retten
 move.w #$2000,d0                ; Farbzähler auf $2000 setzen
```

```
l5:
 move.w d0,$dff180                  ; Farbe aktivieren
 dbra d0,l5                         ; Farbzähler erniedrigen
 movem.l (sp)+,d0-d7/a0-a6          ; Register zurückholen

oldirq:                             ; ab hier nichts ändern (Wichtig)
 jmp oldirq                         ; alte System-IRQ-Routine fort-
                                    ; setzen

;---------------------------- Listing Ende ----------------------
```

4.2 Raster-Interrupt

Ein Rasterinterrupt wird immer dann erzeugt, wenn die letzte Rasterzeile des Bildschirms erreicht wurde. Dieser Interrupt eignet sich besonders gut für Scrolling bzw. ruckfreie Animationen. Außerdem besitzt ein Raster-IRQ eine höhere Priorität als ein Soft-IRQ und wird deswegen nicht von diesem unterbrochen.

Wie man einen solchen Raster-IRQ programmiert, wollen wir anhand von zwei verschiedenen Beispielen demonstrieren.

Die erste wäre im Prinzip genauso wie beim Soft-IRQ. Wir erweitern also nur die System-Raster-IRQ-Routine. Deswegen können wir obiges Programm fast vollständig übernehmen. Es muß lediglich die Ebene 1 durch die Ebene 3 ersetzt werden.

Hier nochmal zur Verdeutlichung das abgeänderte Listing:

```
;-------------------- Programmname = RasterIRQ --------------------
;------------------- wartet auf rechte Maustaste -------------------
;------------------- läßt den Bildschirm blinken -------------------


start:
 move.l $6c,oldirq+2      ; alte IRQ-Adresse auslesen und speichern
 move.l #rasterirq,$6c    ; eigene IRQ-Adresse in Ebene 3 eintragen

wait:
 btst #10,$dff016                ; wurde rechte Maustaste gedrückt
 bne wait                        ; wenn nicht dann weiter abfragen

exit:
 move.l oldirq+2,$6c             ; alte IRQ-Adresse wieder zu-
                                 ; rückschreiben
 rts                             ; Programm beenden

;--------- Hier beginnt Raster-IRQ-Routine ---------

rasterirq:
 movem.l d0-d7/a0-a6,-(sp)       ; Register retten
 move.w #$2000,d0                ; Farbzähler auf $2000 setzen

l5:
 move.w d0,$dff180               ; Farbe aktivieren
 dbra d0,l5                      ; Farbzähler erniedrigen

 movem.l (sp)+,d0-d7/a0-a6       ; Register zurückholen

oldirq:                          ; ab hier nichts ändern (Wichtig)
 jmp oldirq                      ; alte System-IRQ-Routine fort-
                                 ; setzen
;------------------------- Listing Ende -------------------------
```

Ihnen ist bestimmt sofort beim Testen des Programmes aufgefallen, daß die Farbschleife wesentlich schneller abläuft, als beim Soft-IRQ. Dieses liegt, wie oben schon kurz erwähnt, an der höheren Priorität.

Die zweite Möglichkeit einen Raster-IRQ zu erzeugen, ist die über das System. Denn das System stellt uns zur Interruptprogrammierung einige Routinen zur Verfügung. Diese Routinen sind zwar nicht sehr leistungsfähig, wir wollen aber trozdem auf diese Möglichkeit kurz eingehen.

Diese Routinen befinden sich in der Exec-Bibliothek. Und zwar AddIntServer () und RemIntServer ().

Die Routine AddIntServer () fügt einen Interrupt zum System hinzu und die Routine RemIntServer () entfernt diesen wieder.

Die Routinen verlangen lediglich in D0 die Interruptnummer, welche mit den Pseudoprioritäten (siehe obige Tabelle) übereinstimmen und in A1 einen Zeiger auf eine Interruptstruktur. Es können jedoch nicht alle Nummer eingesetzt werden. Denn diese Routinen verwalten nur sogenannte Interrupt-Server-Listen, welche schon vorher vom System initialisiert worden sind.
Mit diesen Listen können mehere Interrupts nacheinander vom System abgearbeitet werden.

Folgende Nummern sind erlaubt: 3, 5, 13 und 15 (wird jedoch nicht vom System benutzt)

Die Interruptstruktur hat folgenden Aufbau:

Interrupt-Struktur: (Länge = 22 Bytes)

Offset:	Typ:	Bezeichnung:	Beschreibung:
000	Long	Succ	Zeiger auf nächste Liste (un wichtig)
004	Long	Pred	Zeiger auf vorangegangene Li ste (unwichtig)
008	Byte	Type	kodierter Node-Typ (auf Wert 2 setzen)
009	Byte	Pri	Priorität dieser Liste (von +127 bis -128)
010	Long	Name	Zeiger auf Namen dieser Liste (unwichtig)
014	Long	Data	Zeiger auf Datenspeicher (un wichtig)
018	Long	Code	Zeiger auf Interruptroutine
022		END	Ende der Interrupt-Struktur

In Code trägt man die Anfangsadresse seiner IRQ-Routine ein, welche ausgeführt werden soll. Diese Routine muß allerdings mit RTS enden.

Beschreibung der Interruptroutinen:

```
Routine   :  AddIntServer (Nummer,Interrupt) (D0,A1)
Library   :  exec.library
Offset    :  -168 = -$A8
Parameter:   D0 =  Interruptnummer - welcher IRQ benutzt werden soll
                   (Erlaubt sind: 3, 5, 13, 15)
             A1 = Zeiger auf Interrupt-Struktur (siehe oben)
Rückgabe :   keine
Erklärung:   Diese Routine fügt einen Interrupt zum System hinzu
```

```
Routine   : RemIntServer (Nummer,Interrupt) (D0,A1)
Library   : exec.library
Offset    : -174 = -$AE
Parameter: D0 =  Interruptnummer - welcher IRQ entfernt werden soll
                 (Erlaubt sind: 3, 5, 13, 15)
           A1 = Zeiger auf Interrupt-Struktur (siehe oben)
Rückgabe : keine
Erklärung: Diese Routine entfernt einen Interrupt wieder aus dem
           System
```

Auf diese Art einen Interrupt zu erzeugen, ist wohl ein wenig komplizierter, jedoch arbeitet sie einwandfrei zusammen mit dem System.

Das folgende Programm erzeugt wie obiges einen Rasterinterrupt, jedoch diesmal über die Systemroutinen.

```
;----------------- Programmname = RasterIRQ(Server) ---------------
;-------------------- wartet auf rechte Maustaste -----------------
;--------------------- läßt Bildschirm blinken --------------------

 addintserver = -168
 remintserver = -174

start:
 move.l #interrupt,a1              ; Zeiger auf Interrupt-Struktur
 move.l #5,d0                      ; IRQ-Nummer (5 = RasterIRQ)
 move.l 4,a6                       ; Execbasis
 jsr addintserver(a6)              ; Interrupt zum System hinzufüh
                                   ; ren
```

```
wait:
 btst #10,$dff016                ; rechte Maustaste gedrückt?
 bne wait                        ; wenn nicht, dann weiter abfra-
                                 ; gen
exit:
 move.l #interrupt,a1            ; Zeiger auf Interrupt-Struktur
 move.l #5,d0                    ; IRQ-Nummer (5 = RasterIRQ)
 move.l 4,a6                     ; Execbasis
 jsr remintserver(a6)            ; Interrupt wieder entfernen
 rts                             ; Programm beenden

;---------- ab hier beginnt Raster-IRQ-Routine ----------

rasterirq:
 movem.l d0-d7/a0-a6,-(sp)       ; Register retten
 move.w #$2000,d0                ; Farbzähler auf $2000 setzen

l5:
 move.w d0,$dff180               ; Farbe aktivieren
 dbra d0,l5                      ; Farbzähler erniedrigen
 movem.l (sp)+,d0-d7/a0-a6       ; Register zurückholen
 rts                             ; mit RTS beenden (Wichtig)

;---------- Parameter ----------

interrupt:                       ; ab hier Interruptstruktur
 dc.l 0,0
 dc.b 2,0                        ; Typ = Interrupt, Pri = Null
 dc.l 0,0
 dc.l rasterirq                  ; Code (Zeiger auf IRQ-Routine)

;-------------------------- Listingende --------------------------
```

Welche von denen hier beschriebenen Möglichkeiten man nun benutzt, um einen Interrupt zu erzeugen, bleibt letzendlich Ihnen überlassen. Jede Routine hat Ihre Vor- bzw. Nachteile.

5. Kapitel

Fonts - Zeichensätze

- Fonts - Zeichensätze
- Aufbau der Fonts
- Aktivieren von Fonts
- Texte ausgeben
- Textroutinen
- Laufschriften

5. Fonts - Zeichensätze

Auf fast jeder Amiga-Diskette befinden sich im Fonts-Verzeichnis (dir) irgend welche Zeichensätze. Man kann sich somit ziemlich schnell und einfach eine kleine Fontsammlung auf einer Extra-Disk anlegen! Oder man kreiert sich, mit einen der zahlreichen Font-Editoren, einen absolut neuen Zeichensatz.

Doch wie bekommt man nun einen solchen Font auf den Bildschirm?

Dazu befindet sich auf der Workbench-Disk im 'Libs-Dir' eine Bibliothek, die sich 'diskfont.library' nennt. Mit der lassen sich ziemlich einfach Zeichensätze in den Speicher laden (öffnen). Und wenn dann erst einmal ein Font ordnungsgemäß geöffnet wurde, stellt die Grafikbibliothek einige Routinen zur Verfügung, mit denen man den Font verwalten kann.

Nach einschalten des Amigas, befinden sich schon zwei Zeichensätze im Speicher. Es sind der "topaz 8" und "topaz 9". Welcher der beiden Zeichensätze nun aktiv ist, hängt von dem eingestelltem Zeichen-Modus in den Preferneces ab (60 bzw. 80 Zeichenmodus).

5.1 Aufbau der Fonts

Wenn man einen Zeichensatz öffnet, erhält man in D0 immer einen Zeiger auf eine TextFont-Struktur zurück. Die Anfangsadresse dieser Struktur wird dann, durch aufrufen entsprechender Routinen, in dem gewünschten RasterPort (Offset 52) eingetragen. Diese TextFont-Struktur braucht das System um einen Zeichensatz auf dem Amiga verwalten zu können.

Aufbau der Datenstruktur "TextFont":

TextFont-Struktur: (Länge = 52 Bytes)

Offset	Typ	Bezeichnung	Beschreibung
000	Long	Succ	Zeiger auf nächsten Font
004	Long	Pred	Zeiger auf vorrangegangenen Font
008	Byte	Type	Node-Typ (12 = Font)
009	Byte	Pri	Priorität
010	Long	Name	Zeiger auf Fontnamen
014	Long	ReplyPort	Zeiger auf AntwortPort
018	Word	Lenght	Länge der Message
020	Word	YSize	Höhe des Zeichensatzes
022	Byte	Style	Stil des Zeichensatzes:
		Normal	normal = 0
		Underline	unterstrichen = 1
		Bold	fett = 2
		Italic	kursiv = 4
			extended = 8
023	Byte	Flags	Preferences und Flags:
			ROM-Font = 1
			Disk-Font = 2
			Rev-Path = 4
			TallDot = 8
			WideDot = 16
			Proportional = 32
			Designed = 64
			Removed = 128
024	Word	XSize	durchschnittliche Fontbreite
026	Word	Baseline	Fonthöhe ohne Unterlängen
028	Word	BoldSmear	Smear-Effekt für Fettdruck
030	Word	Accessors	Zugriffszähler (Null = Font wird gelöscht)

032	Byte	LoChar	ASCII-Code des ersten Zei chens
033	Byte	HiChar	ASCII-Code des letzten Zei chens
034	Long	CharData	Zeiger auf Zeichensatzdaten
038	Word	Modulo	Bytes pro Zeichensatz-Zeile
040	Long	CharLoc	Zeiger auf Offset-Daten für Zeichendecodierung
044	Long	CharSpace	Zeiger auf Breiten-Tabelle der Zeichen
048	Long	CharKern	Zeiger auf Zeichenkern der Ta belle
052		END	Ende der TexFont-Struktur

Erläuterung der Datenstruktur:

Offset 0 bis 20: Message

Damit auf einen Font mehrere Tasks gleichzeitig zugreifen können, bedarf es einer Message-Struktur welche das Multi-Tasking steuert. Für den User Unwichtig.

Offset 20: YSize

Enthält die Höhe des Fonts. Die Höhe ist für alle Zeichen gleich.

Offset 22: Style

Hier steht die Charakteristika des Zeichensatzes. Also wie der Font auf dem Bildschirm ausgegeben werden soll.

Offset 23: Flags

Beschreibt den augenblicklichen Status des Zeichensatzes. Ob er sich z.B. im RAM oder auf der Diskette befindet.

Offset 24 und 26: XSize und Baseline

Weitere Demensionen des Fonts.

Offset 28: BoldSmear

Der Amiga gibt beim Fettdruck ein Zeichen zweimal aus, jedoch beim zweitenmal um einen Punkt nach rechts verschoben. Durch eintragen eines anderen Wertes in BoldSmear, kann das Zeichen auch um größere Abstände verschoben werden.

Offset 30: Accessors

Hier befindet sich die Anzahl der Tasks, die auf diesen Font zugreifen. Wenn ein Task diesen Zeichensatz schließt, wird dieser Zähler um eins erniedrigt. Erst wenn Accessors den Wert Null enthält, wird der Font vollständig aus dem Speicher gelöscht.

Offset 32 und 33: LoChar und HiChar

Der Amiga ist in der Lage bis zu 256 Zeichen in einem Zeichensatz unterzubringen. Jedoch nutzt nicht jeder Zeichensatz diese Kapazität aus. Deswegen werden hier der Wert des ersten und letzten ASCII-Codes eingetragen.

Offset 34: CharData

Hier befindet sich der Zeiger auf die Definition der Zeichen. Das heißt hier liegt die Anfangsadresse der Grafikdaten (des ersten Zeichens).

Offset 38: Modulo

Hier steht die Anzahl Bytes, die eine CharData-Zeile lang ist. Um nun von eine CharData-Zeile in die nächste zu kommen, addiert man lediglich den Modulowert zur augenblicklichen Datenzeile.

Offset 40: CharLoc

Da der Amiga einen Zeichensatz zeilenweise speichert, also erst die erste Zeile aller Zeichen nacheinander, dann die zweite usw. und jeder Buchstabe unterschiedlich breit sein kann, benötigen wir Informationen, wie breit das einzelne Zeichen und an welcher Stelle es im Speicher zu finden ist. Dazu bekommt jedes Zeichen zwei Words zugeteilt. Im ersten steht der Offset zur Anfangsadresse (CharData), im zweiten die Breite in Bits des Zeichens. Diese Zwei-Words-Arrays werden fr jedes Zeichen der Reihe nach abgelegt. CharLoc zeigt dann auf die Anfangsadresse der ersten beiden Words des ersten Zeichens. Dieses gilt jeweils für eine Zeile des Zeichensatzes. Wie kommt man aber von einer CharData-Zeile in die nächste? Siehe dazu Modulobeschreibung.

Offset 44: CharSpace

Das zweite Word in CharLoc enthält zwar die Breite eines Zeichens. Aber stellen Sie sich mal vor, wie es aussieht, wenn man einen Text so ausgeben würde! Der würde absolut gequetscht werden. Deswegen ist jedes Zeichen immer ein paar Punkte breiter als normal. Die tatsächliche Breite eines Zeichens wird in einem

Word gespeichert. Diese Words werden für alle Zeichen nacheinander im Speicher abgelegt. CharSpace zeigt nun auf das erste Word eines Zeichens.

Offset 48: CharKern

Das zweite Word auf das CharLoc zeigt enthält ja die Breite in Bits eines Zeichens und das Word auf das CharSpace zeigt die tatsächliche Breite. Nehmen wir mal an ein Zeichen ist 10 Bits breit und die tatsächliche Breite beträgt 15 Bits. Es stellt sich jetzt die Frage, ab welchen Bit nun die 10 Bits ausgegeben werden sollen? Dafür exestiert ein weiteres Word. In diesem Word ist die Anzahl Bits gespeichert, die das Zeichen vor der Ausgabe nach rechts verschoben werden muß. Auch hier werden für jedes Zeichen ein Word nacheinander im Speicher abgelegt. CharKern zeigt dann auf das erste Word des ersten Zeichens.

5.2 Aktivieren von Fonts

Wenn man einen Zeichensatz aus dem Libs-Verzeichnis in den Speicher laden und danach aktivieren möchte, müssen folgende Schritte durchgeführt werden:

1. "diskfont.library" öffnen
2. Aus dieser Library die Routine "OpenDiskFont ()" aufrufen
3. "graphics.library" öffnen
4. Aus dieser Library die Routine "SetFont" aufrufen

Durch folgende Schritte entfernt man einen Font wieder (denn jeder Font verbraucht Speicherplatz, den man, wenn man sein Programm beenden will, zurückgeben sollte):

1. "graphics.library" öffnen, wenn sie nicht schon geöffnet wurde
2. Aus dieser Library die Routine "CloseFont ()" und "RemFont ()" aufrufen

Hier nun die Beschreibung der einzelnen Routinen:

```
Routine   :  OpenDiskFont (TextAttr) (A0)
Library   :  diskfont.library (befindet sich auf Disk - Workbench)
Offset    :  -30 = -$1e
Parameter:   A0 =  Zeiger auf eine Text-Attribut-Struktur, welche den
                   gewünschten Font genauer beschreibt. (siehe unten)
Rückgabe :   D0 =  Zeiger auf TextFont-Struktur (siehe Aufbau der
                   Fonts), die zu diesem Font gehört, oder Null wenn
                   Error aufgetreten ist.
Erklärung:   Diese Routine lädt einen Font von Disk in den Speicher
             und legt die dazugehörige TextFont-Struktur an.
```

```
Routine   :  SetFont (RastPort,TextFont) (A1,A0)
Library   :  graphics.library
Offset    :  -66 = -$42
Parameter:   A1 =  Zeiger auf eine Rasterportstruktur in dem der ge-
                   wünschte Font aktiviert werden soll.
             A0 =  Zeiger auf eine TextFont-Struktur, den wird durch
                   die Routine "OpenDiskFont" in D0 erhalten haben
Rückgabe :   keine
Erklärung:   Diese Routine aktiviert einen Font im entsprechenden Ra-
             sterport.
```

```
Routine  :  CloseFont (TextFont) (A1)
Library  :  graphics.library
Offset   :  -78 = -$4e
Parameter:  A1 =  Zeiger auf eine TextFont-Struktur, den wird durch
                  die Routine "OpenDiskFont" in D0 erhalten haben
Rückgabe :  keine
Erklärung:  Diese Routine schließt einen Font wieder. Erst wenn ein
            Font geschlossen wurde, kann der Speicher, den er belegt
            wieder freigegeben werden.
----------------------------------------------------------------------

Routine  :  RemFont (TextFont) (A1)
Library  :  graphics.library
Offset   :  -486 = -$1e6
Parameter:  A1 =  Zeiger auf eine TextFont-Struktur, den wird durch
                  die Routine "OpenDiskFont" in D0 erhalten haben
Rückgabe :  keine
Erklärung:  Diese Routine löscht einen Font aus dem Speicher.
----------------------------------------------------------------------
```

Wenn man einen Zeichensatz von Disk laden möchte, braucht man also lediglich die Routine "OpenDiskFont ()" aufzurufen. An diese Routine muß allerdings eine Struktur (TextAttr) übergeben werden. Diese ist notwendig, da in ihr die nötigen Parameter, welche von Ihnen eingetragen werden, stehen. Diese Struktur ist ziemlich kurz und folgendermaßen aufgebaut:

TextAttr-Struktur: (Länge = 8 Bytes)

Offset	Typ	Bezeichnung	Beschreibung
000	Long	FontName	Zeiger auf Fontnamen, welcher mit Null endet
004	Word	Height	Fonthöhe (steht im jeweiligen Fonts-Dir)

006	Byte	Style	Stil des Fonts (siehe TextFont-Struktur)
007	Byte	Flags	Preferences (siehe TextFont-Struktur)
008		END	Ende der TextAttr-Struktur

Um z.B. den topaz.font von Disk zu laden trägt man in Height die gewünschte Höhe ein (z. B. 8), in Flags den Wert 2 (DiskFont), damit der Font von Disk geladen wird und läßt den Fontname auf den Zeichensatznamen zeigen.

z. B.

```
init_textattr:
 move.l #fontname,textattr          ; Zeiger auf Fontname
 move.w #8,textattr+4               ; Höhe des Fonts
 move.b #2,textattr+7               ; Font von Disk laden
 rts

fontname: dc.b "topaz.font",0
 even
```

5.3 Texte ausgeben

Für das Ausgeben von Texten stellt das System uns zwei Routinen zur Verfügung. Einmal "PrintIText ()", welche wir in der Intuitions-Bibliothek finden und einmal "Text ()", welche uns die Grafik-Bibliothek bereit stellt.

Beide Routinen haben ihre Vor- und Nachteile. Der "PrintIText ()"-Routine kann man gleich die gewünschten Koordinaten und Farbe übergeben. Sie benötigt jedoch eine Struktur und ist etwas umständlich zu handhaben.

Die "Text ()"-Routine braucht keine zusätzliche Struktur und ist auch ein wenig schneller als die obige, dafür geht sie aber immer von der gerade aktuellen Position und Schriftfarbe aus.

Beschreibung der beiden Routinen:

```
Routine   :  PrintIText (RastPort,IText,Left,Top) (A0,A1,D0,D1)
Library   :  intuition.library
Offset    :  -216 = -$d8
Parameter:   A0 =  Zeiger auf RastPort-Struktur in dem der Text dar-
                   gestellt werden soll.
             A1 =  Zeiger auf IText-Struktur, die unteranderem den
                   Text enthält (Aufbau siehe unten)
             D0 =  X-Position des Textes
             D1 =  Y-Position des Textes
Rückgabe :   keine
Erklärung:   Diese Routine gibt einen Text auf dem Bildschirm aus, wie
             er in der IText-Struktur beschrieben wird.
```

Aufbau der IText-Struktur: (Länge = 20 Bytes)

Offset	Typ	Bezeichnung	Beschreibung
000	Byte	DetailPen	TextFarbe (Farbregisternummer)
001	Byte	BlockPen	Hintergrundfarbe (Farbregisternummer)
002	Byte	Mode	Darstellungsart (0, 1, 4, 5, 8)
003	Byte	Pad	unbenutzt

004	Word	x_pos	relative X-Position zur aktuel len Position
006	Word	y_pos	relative Y-Position zur aktuel len Position
008	Long	Font	Zeiger auf TextFont-Struktur oder Null
012	Long	Text	Zeiger auf Text, welcher mit Null endet
016	Long	NextIText	Zeiger auf weitere IText-Struk tur oder Null
020		END	Ende der IText-Struktur

```
Routine   :  Text (RastPort,String,Count) (A1,A0,D0)
Library   :  graphics.library
Offset    :  -60 = -$3c
Parameter:   A1 =  Zeiger auf RastPort-Struktur in dem der Text dar-
                   gestellt werden soll.
             A0 =  Anfangsadresse des auszugebenen Textes.
             D0 =  Anzahl Buchstaben die ausgegeben werden sollen.
Rückgabe :   keine
Erklärung:   Diese Routine gibt eine Anzahl von Buchstaben auf dem
             Bildschirm aus. Allerdings an den aktuellen Koordinaten
             bzw. in der aktuellen Schriftfarbe.
```

Nach soviel Theorie, wollen wir endlich auch mal einen Text in die Praxis umsetzen, sprich auf dem Bildschirm bringen. Wir verwenden dazu einen Font von der Diskette.

Das folgende Programm führt folgende Schritte durch:

1. intuition.library und graphics.libary öffnen
2. diskfont.library öffnen
3. einen Lo-Res Screen öffnen
4. ScreenRastPort ermitteln
5. mit OpenDiskFont () einen Font von Diskette laden
6. mit SetFont () den neuen Font aktivieren
7. mit PrintIText einen Text auf dem Bildschirm ausgeben
8. Warten auf rechte Maustaste
9. Screen wieder schließen

A. mit CloseFont () den neuen Font wieder schließen
B. mit RemFont () den neuen Font aus dem Speicher löschen
C. intuition.library und graphics.library schließen
D. diskfont.library schließen und Programm beenden

```
;---------------------- Programmname = OpenFont -------------------
;------- Lädt Font von Disk und gibt diesen auf Bildschirm aus ------
;-------------------- Wartet auf rechte Maustaste -----------------

;---------- Intuitionlibrary öffnen ----------

 move.l 4,a6                        ; ExecBase
 move.l #intname,a1                 ; Namen
 clr.l d0                           ; Version
 jsr -552(a6)                       ; OpenLibrary
 move.l d0,intbase                  ; Basis speichern

;---------- Grafik-Library öffnen ----------

 move.l 4,a6
 move.l #gfxname,a1
```

```
 clr.l d0
 jsr -552(a6)                        ; OpenLibary
 move.l d0,gfxbase                   ; Basis speichern

;---------- Diskfont-Library öffnen ----------

 move.l 4,a6
 move.l #diskfontname,a1
 clr.l d0
 jsr -552(a6)                        ; OpenLibary
 move.l d0,diskfontbase              ; Basis speichern

;---------- screen öffnen ----------

 move.l intbase,a6
 move.l #newscreen,a0                ; Zeiger auf Screen-struktur
 jsr -198(a6)                        ; OpenScreen
 move.l d0,screenhd

;---------- screenrasterport ermitteln ----------

 move.l screenhd,d0
 add.l #84,d0
 move.l d0,screenrastport

;--------- Font öffnen ----------

 move.l diskfontbase,a6
 move.l #fontstruk,a0                ; Zeiger auf textAttr-Struktur
 jsr -30(a6)                         ; OpenDiskFont
 move.l d0,fontadresse               ; in D0 = zeiger auf textFont-
                                     ; struktur
 cmp.l #0,d0                         ; Fehler?
 beq exit                            ; Wenn ja, dann ende!
```

```
;---------- Neuen Font aktivieren ----------

 move.l gfxbase,a6
 move.l screenrastport,a1          ; Zeiger auf Rasterportstruktur
 move.l fontadresse,a0             ; Zeiger auf textFont-struktur in
                                   ; a0 laden
 jsr -66(a6)                       ; SetFont

;---------- Demo Text Printen ----------

 move.l #10,d0                     ; x-position
 move.l #30,d1                     ; y-position
 move.l #text1,a1                  ; Zeiger auf Text-struktur
 move.l screenrastport,a0          ; Zeiger auf Rasterport-struktur
 move.l intbase,a6                 ; Basis
 jsr -216(a6)                      ; PrintIText

;---------- Auf rechte Maustaste in Port 1 warten ----------

main:
 btst #10,$dff016
 beq exit
 bra main


;---------- Schließen des screens -----------

exit:
 move.l intbase,a6
 move.l screenhd,a0
 jsr -66(a6)                       ; CloseScreen
```

```
;---------- Font wieder schließen ----------

 move.l gfxbase,a6
 move.l fontadresse,a1
 cmp.l #0,a1
 beq exit_1
 jsr -78(a6)                    ; CloseFont

;---------- Font wieder löschen ----------

 move.l gfxbase,a6
 move.l fontadresse,a1
 cmp.l #0,a1                    ; Font überhaupt vorhanden ?
 beq exit_1
 jsr -486(a6)                   ; RemFont

;---------- Librarys schließen ----------

exit_1:
 move.l 4,a6
 move.l intbase,a1
 jsr -414(a6)                   ; CloseLibrary
 move.l 4,a6
 move.l gfxbase,a1
 jsr -414(a6)                   ; CloseLibrary
 move.l diskfontbase,a1
 move.l 4,a6
 jsr -414(a6)                   ; CloseLibrary
 rts

;---------- Parameter ----------

intname: dc.b "intuition.library",0
 even
gfxname: dc.b "graphics.library",0
 even
```

```
diskfontname: dc.b "diskfont.library".0
 even
gfxbase: dc.l 0
intbase: dc.l 0
screenhd: dc.l 0
fontadresse: dc.l 0
diskfontbase: dc.l 0
screenrastport: dc.l 0

;----------- Screen-Struktur ----------

newscreen:
 dc.w 0.0.320.256.5
 dc.b 0.1
 dc.w 0.15
 dc.l 0.title.0.0
title: dc.b "Font-Demo".0
 even

;---------- Text-Struktur ----------

text1:
 dc.b 2                 ; Textfarbe aus colorregister 2
 dc.b 0                 ; Hintergrundfarbe aus colourregister 1
 dc.b 1                 ; inverse Textdarstellung
 dc.b 0                 ; unbenutzt
 dc.w 0                 ; relativer x-abstand
 dc.w 0                 ; relativer y-abstand zu gesetzten Koordina-
                        ; ten
 dc.l 0                 ; Standartzeichensatz
 dc.l texta             ; Zeiger auf Text
 dc.l 0                 ; keine weitere Text-Struktur
texta: dc.b "Der neue Zeichensatz !".0
 even
```

```
;---------- Font-Struktur (textAttr) ----------

fontstruk:
 dc.l fontname                          ; Zeiger auf Fontnamen
 dc.w 20                                ; Font-Höhe = 20
 dc.b 0                                 ; Normal Darstellung
 dc.b 2                                 ; Font auf Disk
fontname: dc.b "diamond.font",0         ; Fontname
 even

;---------------------- Ende des Listings ------------------------
```

5.4 Textroutinen

Wenn man nun die 'Text ()'-Routine aus der graphics.library benutzt, ist es ja nun nicht gerade von Vorteil, da° der Text an einer "zufälligen" Position auf dem Bildschirm oder wohlmöglich in der selben Farbe, wie der Hintergrund erscheint. Was zur Folge hätte, daß man den Text überhaupt nicht sieht. Oder vielleicht möchten Sie den Text auch kursiv oder in Fettdruck darstellen, wie Sie es von Textverarbeitungsprogrammen kennen.

Für diese Dinge und noch einiges mehr, stellt das System einige Routinen zur Verfügung. Diese Routinen finden wir alle in der graphics.library.

Will man z. B. wissen, wie lang ein bestimmter Satz ist, trägt man in A0 die Anfangsadresse des Satzes ein, in D0 die Anzahl Buchstaben und in A1 die RastPort-Adresse von Window bzw. Screen. Sofern diese Schritte ordnungsgemäß durchgeführt worden sind, ruft man die Routine 'TextLength ()' auf und erhält in D0 die Anzahl Punkte zurück, die der Text lang ist. Dieses ist von Vorteil, wenn man wissen will, ob der Text noch auf die Zeile paßt.

Hier ein kleines Beispiel dazu, wie so etwas programmiertechnisch aussieht:

```
 move.l #gfxbase,a6        ; Grafikbasisadresse
 move.l #rastport,a1       ; RastPortadresse (z. B. eigen angelegte)
 move.l #text,a0           ; Anfangsadresse vom Text
 move.l #textende-text,d0  ; Länge des Textes
 jsr -54(a6)               ; TextLength () aufrufen
 move.l d0,textlänge       ; Textlänge speichern
 rts

text: dc.b "Wie lang ist der Text?"
textende:
 even
textlänge: dc.l 0
rastport: blk.b 100,0
```

Eine ausführliche Beschreibung jeder Routine würde hier zu weit führen. Deshalb folgt eine kurze Beschreibung jeder einzelnen Routine. Zudem ist die Kurzbeschreibung wesentlich übersichtlicher als eine in Sätze gefaßte. Anfangen werden wir mit der 'TextLength ()'-Routine:

```
Routine   : TextLength (RastPort,String,Count) (A1,A0,D0)
Library   : graphics.library
Offset    : -54 = -$36
Parameter: A1 =  Zeiger auf RastPort-Struktur in dem der entspre-
                 chende Zeichensatz zu finden ist.
           A0 =  Anfangsadresse des Textes.
           D0 =  Anzahl Buchstaben die der Text besitzt
Rückgabe : in D0 = Anzahl Punkte die der Text lang ist.
Erklärung: Diese Routine gibt in D0 die Anzahl Punkte zurück, die
           der Text lang ist. Die Textlänge bezieht sich immer auf
           die aktuelle Zeichenbreite des aktiven Zeichensatzes im
           übergebenen RastPort.
```

Routine : AskSoftStyle (RastPort) (A1)
Library : graphics.library
Offset : -84 = -$54
Parameter: A1 = Zeiger auf RastPort-Struktur in dem der entsprechende Zeichensatz zu finden ist.
Rückgabe : in D0 = Stylebyte welches den aktuellen Modus des Fonts wiederspiegelt.

Stylename:	Wert:	Bit:	Funktion:
Normal	0	-	normal
Underline	1	0	unterstrichen
Bold	2	1	fett
Italic	4	2	kursiv
	8	3	extended

Erklärung: Diese Routine gibt in D0 den Style-Modus zurück, welcher gerade aktiv ist. Es sind auch mehrere Styls aufeinmal möglich. Man braucht dazu lediglich die einzelnen Werte zusammenaddieren.

Routine : SetSoftStyle (RastPort,Style,Enable) (A1,D0,D1)
Library : graphics.library
Offset : -90 = -$5a
Parameter: A1 = Zeiger auf RastPort-Struktur in dem der entsprechende Zeichensatz zu finden ist.
D0 = Stylebyte welches den aktuellen Modus des Fonts wiederspiegelt.

Stylename:	Wert:	Bit:	Funktion:
Normal	0	-	normal
Underline	1	0	unterstrichen
Bold	2	1	fett
Italic	4	2	kursiv
	8	3	extended

D1 = Enthält den Wert, der Auskunft darüber gibt, welche Styles überhaupt zugelassen sind. Also die Stylemaske. Am besten Sie setzen D1 auf 15.

Erklärung: Diese Routine setzt den Style-Modus im entsprechenden RastPort. Wenn Sie jetzt einen Text in diesem RastPort ausgeben, wird dieser entsprechend anders ausgegeben.

Routine : SetRast (RastPort,Color) (A1,D0)
Library : graphics.library
Offset : -234 = -$ea
Parameter: A1 = Zeiger auf RastPort-Struktur in dem der entsprechende Zeichensatz zu finden ist.
D0 = Farbregisternummer, aus dem die Farbe entnommen werden soll, fürs Zeichnen von Linien oder ausgeben von Texten
Erklärung: Diese Routine setzt die Punktfarbe im entsprechenden RastPort.

Routine : Move (RastPort,X,Y) (A1,D0,D1)
Library : graphics.library
Offset : -240 = -$f0
A1 = Zeiger auf RastPort-Struktur
D0 = Position der X-Koordinate
D1 = Position der Y-Koordinate
Erklärung: Diese Routine setzt die Koordinaten im entsprechenden RastPort. Ab diesen Positionen wird dann der Text bzw. die Linie dargestellt.

Routine : SetAPen (RastPort,Pen) (A1,D0)
Library : graphics.library
Offset : -342 = -$156
Parameter: A1 = Zeiger auf RastPort-Struktur
D0 = Farbregisternummer, aus dem die Schriftfarbe entnommem werden soll.
Erklärung: Diese Routine setzt die Schriftfarbe im entsprechenden RastPort.

Routine : SetBPen (RastPort,Pen) (A1,D0)
Library : graphics.library
Offset : -348 = -$15c
Parameter: A1 = Zeiger auf RastPort-Struktur
D0 = Farbregisternummer, aus dem die Hintergrundfarbe entnommem werden soll.
Erklärung: Diese Routine setzt die Hintergrundfarbe für die Textausgabe im entsprechenden RastPort.

Routine : SetDrMd (RastPort,DrawMode) (A1,D0)
Library : graphics.library
Offset : -354 = -$162
Parameter: A1 = Zeiger auf RastPort-Struktur
D0 = Zeichenmodus für die Textausgabe:

Name:	**Bit:**	**Funktion:**
JAM1	-	Es wird nur in der Farbe des APens gemalt
JAM2	0	Es wird mit der Farbe des BPens unterlegt
Complement	1	Nur gesetzte Punkte werden in Screen aus gegeben
Inversvid	2	Text wird invertiert ausgegeben

Erklärung: Diese Routine setzt den Zeichenmodus für die Textausgabe im entsprechenden RastPort.

5.5 Laufschriften

Wer kennt sie nicht, diese sogenannten 'Intros' bzw. 'Demos', welche meist von Crackern oder Programmierer erstellt werden, um irgend welche 'Messages' unterlegt mit einen super Sound + Bild , in der eine Laufschrift abläuft, herüber zu bringen.

Wie man nun eine solche Laufschrift programmiert, darauf wollen wir hier eingehen. Allerdings werden wir hier nur die Programmierung einer ganz einfachen Laufschrift beschreiben.

Als erstes müssen wir uns einmal überlegen, was wir so alles für eine Laufschrift benötigen?

Da ist zunächstmal der Text, den ohne Text keine Laufschrift. Da der Text ja von rechts hereinscrollen soll, benötigen wir eine Routine die in der Lage ist einen Buchstaben auszugeben. Wir verwenden dazu die 'Text ()'-Routine. Der neu eingefügte Buchstabe soll ja immer an der selben Position dargestellt werden. Dies erledigt die Routine 'Move ()' welche von uns festeingestellte Werte übergeben bekommt.

Jedoch wie lassen wir nun den Text über den Bildschirm scrollen?

Selbst dafür stellt uns das System eine Routine zur Verfügung. Sie nennt sich 'ScrollRaster ()'. Es folgt die Beschreibung der Routine:

```
Routine   :  ScrollRaster (RastPort,DX,DY,MinX,MinY,MaxX,MaxY) (A1,D0
             bis D5)
Library   :  graphics.library
Offset    :  -396 = -$18c
Parameter:   A1 =  Zeiger auf RastPort-Struktur
             D0 =  Anzahl X-Punkte um die gescrollt werden soll
```

D1 = Anzhal Y-Punkte um die gescrollt werden soll
D2 = Anfangs X-Koordinate
D3 = Anfangs Y-Koordinate
D4 = End X-Koordinate
D5 = End Y-Koordinate
Erklärung: Diese Routine scrollt den angegebenen Bereich um die entsprechende Anzahl von Punkten

Wir stellen in den Datenregistern D2 bis D5 den Bereich ein der gescrollt werden soll und in D0 eine Eins und D1 eine Null, für Scrolling nach Links um einen Punkt. Jetzt sind wir zwar in der Lage, den Text zu scrollen, aber wann erscheint nun der nächste Buchstabe?

Ganz einfach! Wir ermitteln einfach mit der Routine 'TextLength()' die Länge des momentanen Zeichens. Dann zählen wir einfach eine Variable bis zum ermittelten Wert herauf, wenn dieser erreicht ist, printen wir den nächsten Buchstaben und setzen die Variable wieder auf Null.

Jetzt stehen wir allerdings noch vor einem Problem. Denn in der momentanen Version würde jeder neue Buchstabe sofort hingeprintet werden. Diesen Effekt kann man bei sehr vielen Programmen feststellen. Doch wir wollen uns nicht mit so einer primitiven Art von Laufschrift abgeben, sondern das Zeichen richtig von rechts nach links hereinscrollen lassen.

Da wir unsere Zeichen in einem RastPort darstellen und alles was nicht innerhalb diesen RastPorts liegt, wird vom System einfach abgeschnitten. Diese Tatsache machen wir uns einfach zu nutze und printen das aktuelle Zeichen immer an der letztmöglichen Position des RastPort minus der Anzahl von Punkten die schon gescrollt wurden, bis das nächste Zeichen erscheint.

Damit der Text auch wieder von Vorne beginnt, fragen wir einfach ab, ob das Textende schon erreicht wurde, wenn ja dann lassen wir den Textpointer einfach wieder auf den Anfang des Textes zeigen.
Wenn wir die Laufschrift wie oben beschrieben programmieren würden, wäre sie viel zu schnell und würde außerdem noch flakkern. Deswegen lassen wir die Laufschrift im Interrupt ablaufen.

Hier nun das ausführlich dokumentierte Beispiellisting, was eine Laufschrift auf dem Screen erzeugt:

```
;-------------------- Programmname = Laufschrift --------------------
;--------------------- erzeugt eine Laufschrift ---------------------
;-------------------- rechte Maustaste = Prgende --------------------

start:
;---------- Graphics.library öffnen ----------

 move.l #gfxname.a1                        ; Libraryname
 clr.l d0                                  ; Version
 move.l 4.a6                               ; Execbase
 jsr -552(a6)                              ; OpenLibary
 move.l d0.gfxbase                         ; Basis speichern

;---------- Intuitions.library öffnen ----------

  move.l #intname.a1                       ; Libraryname
  clr.l d0                                 ; Version
  move.l 4,a6                              ; Execbase
  jsr -552(a6)                             ; Openlibrary
  move.l d0,intbase                        ; Basis speichern
```

```
;---------- Screen öffnen ----------

 move.l intbase,a6                      ; Intuitionbasis
 move.l #screenargs,a0                  ; ScreenArgs-Struktur
 jsr -198(a6)                           ; OpenScreen
 move.l  d0,screenhd                    ; Screenadresse speichern

;---------- Fenster öffnen ----------

 move.l intbase,a6                      ; Intuitionbasis
 move.l #windowargs,a0                  ; WindowArgs-Struktur
 jsr -204(a6)                           ; OpenWindow
 move.l d0,windowhd                     ; Windowadresse speichern

;---------- RastPort-Adresse vom Window ermitteln ----------

 move.l windowhd,a1                     ; Windowadresse
 move.l 50(a1),rastport                 ; RasterPort-Adresse vom Window
                                        ; speichern

;---------- IRQ einschalten ----------

 move.l 4,a6                            ; Execbasis
 jsr -132(a6)                           ; Forbid Multitasking
 move.l $6c,irqold+2                    ; alten IRQ speichern
 move.l #irqnew,$6c                     ; neuen IRQ aktivieren

main:
 btst #10,$dff016                       ; rechte Maustaste gedrückt ?
 beq exit                               ; wenn ja, dann Ende
 bra main
```

```
;---------- IRQ wieder entfernen ----------

exit:
 move.l irqold+2,$6c              ; alten IRQ aktivieren
 move.l 4,a6                      ; Execbasis
 jsr -138(a6)                     ; Permit Multitasking

;---------- Window wieder schließen ----------

 move.l intbase,a6                ; Intuitionbasis
 move.l windowhd,a0               ; Windowadresse
 jsr -72(a6)                      ; CloseWindow

;---------- Screen wieder schließen ----------

 move.l intbase,a6                ; Intuitionbasis
 move.l screenhd,a0               ; Screenadresse
 jsr -66(a6)                      ; CloseScreen

;---------- Intuition.library wieder schließen ----------

 move.l intbase,a1                ; Intbase
 move.l 4,a6                      ; Execbasis
 jsr -414(a6)                     ; CloseLibrary

;---------- Graphics.library wieder schließen ----------

 move.l gfxbase,a1                ; graphicsbase
 move.l 4,a6                      ; Execbase
 jsr -414(a6)                     ; CloseLibrary
 rts                              ; Programmende
```

```
;---------- Neue IRQ-Routine in der die Laufschrift erzeugt wird -

irqnew:
 movem.l d0-d7/a0-a6,-(sp)          ; Register retten
 move.l rastport,a1                 ; RastPort-Adresse vom Window
 move.l #1,d0                       ; Anzahl x-punkte  (= Eins)
 move.l #0,d1                       ; Anzhal y-punkte  (= Null)
 move.l #0,d2                       ; Anfangs X-Koordinate
 move.l #0,d3                       ; Anfangs Y-Koor.
 move.w windowargs+4,d4             ; End X-Koordinate
 move.l #70,d5                      ; End Y-Koor.
 move.l gfxbase,a6                  ; Graphics.library
 jsr -396(a6)                       ; ScrollRaster
 move.l rastport,a1                 ; RastPort-Adresse vom Window
 move.l scrollziffer,a0             ; Zeiger auf Text
 move.l #1,d0                       ; einen Buchstaben
 move.l gfxbase,a6                  ; graphicsbasis
 jsr -54(a6)                        ; TextLength
 move.w d0,textlänge                ; Zeichenlänge speichern

 add.w #1,scrollbit                 ; Scrollzähler um eins erhöhen
 move.w scrollbit,d0                ; Scrollvariable nach D0
 cmp.w textlänge,d0                 ; Scrollvariable >= Zeichenbreite

 bge next_ziffer                    ; Wenn ja, nächstes Zeichen

 move.w windowargs+4,d0
 sub.w #1,d0                        ; max. x-Position für MOVE-Rou-
                                    ; tine
 sub.w scrollbit,d0                 ; minus Scrollvariable
 bra printziffer                    ; und Zeichen printen

next_ziffer:                        ; nächstes Zeichen printen
 move.w windowargs+4,d0
 sub.w #1,d0                        ; max. x-position für Move-Rou-
                                    ; tine
```

```
 clr.w scrollbit                    ; Scrollvariable löschen
 add.l #1,scrollziffer              ; Textpointer um eins erhöhen

 cmp.l #scrolltextende,scrollziffer ; schon Textende erreicht ?
 bne printziffer                    ; wenn nein, dann weiter
 move.l #scrolltext,scrollziffer    ; sonst Textpointer auf Textan-
                                    ; fang

printziffer:                ; ein Zeichen ausgeben (D0 wird oben ermit-
                            ; telt)
 move.l rastport,a1         ; Rasterportadresse vom Window
 move.l #60,d1              ; Y-Position des Textes, der gescrollt wird
 move.l gfxbase,a6          ; graphicsbasis
 jsr -240(a6)               ; MOVE  - Koordinaten setzen
 move.l rastport,a1         ; Rasterportadresse vom Window
 move.l scrollziffer,a0     ; Zeiger auf Text, der gescrollt wird..
 move.l #1,d0               ; einen Buchstaben
 move.l gfxbase,a6          ; graphicsbasis
 jsr -60(a6)                ; TEXT  - Zeichen auf Screen darstellen

 movem.l (sp)+,d0-d7/a0-a6          ; Register zurückholen
irqold:
 jmp irqold                         ; alte IRQ-Routine fortsetzen


;---------- Parameter ----------

screenargs:
x_pos:       dc.w    0
y_pos:       dc.w    0
width:       dc.w  320
height:      dc.w  256
depth:       dc.w    5
detail_pen:  dc.b    1
block_pen:   dc.b    0
view_modes:  dc.w    2
```

```
screen_type:dc.w   $100              ; Normal = 15 für Title
font:       dc.l    0
title:      dc.l    0
gadgets:    dc.l    0
bitmap:     dc.l    0

windowargs:
  dc.w   50
  dc.w   80
  dc.w  250
  dc.w   70
  dc.b    5
  dc.b    0
  dc.l    0
  dc.l $0d00
  dc.l    0
  dc.l    0
  dc.l    0

screenhd:
  dc.l    0
  dc.l    0
  dc.w  50
  dc.w  10
  dc.w  320
  dc.w  256
  dc.w   15

textlänge: dc.w 0
rastport: dc.l 0
windowhd:  dc.l  0
intname: dc.b "intuition.library",0
 even
intbase: dc.l 0
gfxname: dc.b "graphics.library",0
 even
```

```
gfxbase: dc.l 0
scrollbit: dc.w 0                    ; Variable für Anzahl Punkte
scrollziffer: dc.l scrolltext        ; Zeiger auf ScrollText
scrolltext: dc.b "Hallo ! - Dies ist eine einfache "
            dc.b "Scrolldemo! ........."
scrolltextende:
 even

;---------------------------- Listingende --------------------------
```

6. Kapitel

Verbindung zur Außenwelt

- Verbindung zur Außenwelt
- Joystick und Maus
- Tastatur

6. Verbindung zur Außenwelt

In diesem Kapitel werden wir ausnahmsweise auf die Hardwareprogrammierung eingehen. Denn es ist einfacher, schneller und kürzer die Maus bzw. die Tastatur über die Hardware abzufragen, als über das System. Außerdem stellt das System keinerlei Hilfen zur Joystickprogrammierung zur Verfügung.

6.1 Joystick und Maus

Die Hardware des Amiga stellt uns für die Gameports zwei Register zur Verfügung, in denen sich die Gameportsignale wiederspiegeln. Es sind wie alle Register im Amiga 16-Bit-Register, also ein Word groß. Das Register für Gameport 0 liegt bei der Adresse $DFF00A (Joy0Dat) und für Gameport 1 bei Adresse $DFF00C (Joy1Dat).

Beide besitzen die selbe Bitbelegung und sind nur Lese-Register:

```
Bit-Nr.:     15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
Funktion:    Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0 X7 X6 X5 X4 X3 X2 X1 X0
```

Y0 bis Y7 = Y-Zähler (Vertikale Mausdifferenz)
X0 bis X7 = X-Zähler (Horizontale Mausdifferenz)

Sie werden sich sicherlich fragen, was diese Register mit der Mausdifferenz zu schaffen haben? Dies liegt daran, das die Maus und der Joystick über das selbe Register abgefragt werden. In diesen Register spiegelt sich immer die Anzahl Punkte wieder um die die Maus bewegt worden ist. Deswegen muß die Maus immer in bestimmten Zeitabständen abgefragt werden (z. B. im Interrupt). Man vergleicht dann denn alten Zählerstand, welchen man vorher

zwischengespeichert hat, mit dem neuen und die Differenz daraus sind die Anzahl Punkte um die die Maus bewegt worden ist.

Um jetzt den Joystick abzufragen, müßte man das genaugenommen kompliziert über bestimmte Bitverknüpfungen bewerkstelligen. Doch zum Glück exestiert da noch eine zweite Möglichkeit, die erheblich einfacher ist und zudem alle möglichen Kombinationen (einschließlich Feuerknopf) umfaßt.

Wenn wir den Joystick in eine Richtung drücken, enthält das entsprechende Portregister einen bestimmten Wert. Wenn man sich die max. 8 möglichen Werte notiert hat, braucht man diese nur mit dem Portregister zu vergleichen. Wenn man allerdings den Joystick so Richtig umrührt, entstehen ab und zu ganz andere Werte als Vorher und die Joystickabfragung würde nicht mehr funktionieren. Damit das nicht eintreten kann setzen wir zu Beginn jeder Joystickabfragung das JOYTEST-Register, welches bei Adresse $DFF036 liegt auf Null. Der Wert den wir hier eintragen wird nähmlich in die Portregister übertragen.

Bevor wir jetzt alle möglichen Kombinationen des Joysticks programmieren können, fehlt uns noch das Wissen über die Abfragung der Maus bzw. Feuer-Tasten.

Abfragung der Maus/Feuer-Tasten:

```
rechte (Port0):   Loop: BTST #10,$DFF016
                        BNE Loop
linke (Port0)
und Feuerknopf:   Loop: BTST #6,$BFE001
                        BNE Loop

rechte (Port1):   Loop: BTST #14,$DFF016
                        BNE Loop

linke (Port1)
```

```
und Feuerknopf:  Loop: AND.B #128,$BFE001
                       BNE Loop
```

Hier nun ein Beispiellisting welches die Joystickprogrammierung demonstriert. Wenn Sie die Routine in Ihren Programmen einsetzen, sollten Sie alle überflüssigen Abfragungen entfernen, um ein einwandfreies Ablaufen zu garantieren.

```
;--------------- Joystickabfrage (Port 1) als Unterroutine ---------

Port2:
 move.w #0,$dff036       ; GamePortsignale auf Null (Sehr Wichtig!!!)
 move.w $dff00c,d0       ; Wert von GamePort1 nach D0

; ---------- oder $DFF00A für Gameport 0 ----------

 cmp.w #$0100,d0                    ; Oben
 beq P2_oben
 cmp.w #$0001,d0                    ; Unten
 beq P2_unten
 cmp.w #$0003,d0                    ; Rechts
 beq P2_rechts
 cmp.w #$0300,d0                    ; Links
 beq P2_links
 cmp.w #$0103,d0                    ; Oben/Rechts
 beq P2_oben_rechts
 cmp.w #$0002,d0                    ; Unten/Rechts
 beq P2_unten_rechts
 cmp.w #$0301,d0                    ; Unten/Links
 beq P2_unten_links
 cmp.w #$0200,d0                    ; Oben/links
 beq P2_oben_links
 and.b #128,$bfe001                 ; Fire
 beq P2_fire
 rts
```

```
P2_rechts:
 and.b #128,$bfe001
 beq P2_fire_rechts
;---------- Ab hier Routine für Rechts ----------
 rts

P2_links:
 and.b #128,$bfe001
 beq P2_fire_links
;---------- Ab hier Routine für Links ----------
 rts

P2_oben:
 and.b #128,$bfe001
 beq P2_fire_oben
;---------- Ab hier Routine für Oben -----------
 rts

P2_oben_rechts:
 and.b #128,$bfe001
 beq P2_fire_oben_rechts
;---------- Ab hier Routine für Oben/Rechts ----------
 rts

P2_unten_rechts:
 and.b #128,$bfe001
 beq P2_fire_unten_rechts
;---------- Ab hier Routine für Unten/Rechts ----------
 rts

P2_unten_links:
 and.b #128,$bfe001
 beq P2_fire_unten_links
;---------- Ab hier Routine für Unten/Links ----------
 rts
```

```
P2_oben_links:
 and.b #128,$bfe001
 beq P2_fire_oben_links
;---------- Ab hier Routine für Oben/Links ----------
 rts

P2_unten:
 and.b #128,$bfe001
 beq P2_fire_unten
;---------- Ab hier Routine für Unten ----------
 rts

P2_fire:
;---------- Ab hier Routine für Fire ----------
 rts

P2_fire_rechts:
;---------- Ab hier Routine für Fire/Rechts ----------
 rts

P2_fire_links:
;---------- Ab hier Routine für Fire/Links ----------
 rts

P2_fire_oben:
;---------- Ab hier Routine für Fire/Oben ----------
 rts

P2_fire_unten:
;---------- Ab hier Routine für Fire/Unten ----------
 rts

P2_fire_oben_rechts:
;---------- Ab hier Routine für Fire/Oben/Rechts ----------
 rts
```

```
P2_fire_unten_rechts:
;---------- Ab hier Routine für Fire/Unten/Rechts ----------
 rts

P2_fire_unten_links:
;---------- Ab hier Routine für Fire/Unten/Links ----------
 rts

P2_fire_oben_links:
;---------- Ab hier Routine für Fire/Oben/Links ----------
 rts
```

6.2 Tastatur

Für die Tastaturabfrage gibt es im Amiga auch ein Register. Es liegt bei der Adresse $BFEC01 und enthält immer den gerade aktuellen Tastencode. Die Tastencodes stimmen aber nicht mit den ASCII-Codes, wie sie im Basic-Handbuch stehen überein. Die folgende Tabelle gibt Auskunft über die Tastencodes mit der dazugehörigen Tastaturbelegung:

Tastaturabfrage: (Register $BFEC01)

Wert:	**Taste:**	**Wert:**	**Taste:**	**Wert:**	**Taste:**
BF	A	EB/E1	Null	7D	Pfeil-Links
95	B	FD/C5	Eins	73	Del
99	C	FB/C3	Zwei	7B	Tabulator
BB	D	F9/C1	Drei	FF	•
DB	E	F7/A5	Vier	3B	Caps-Lock
B9	F	F5/A3	Fünf	3F	Shift links
B7	G	F3/A1	Sechs	3D	Shift rechts
B5	H	F1/85	Sieben	39	Control

D1	I	EF/83	Acht	37	Alt links
B3	J	ED/81	Neun	35	Alt rechts
B1	K	67	Cursor up	33	Amiga links
AF	L	65	Cursor down	31	Amiga rechts
91	M	61	Cursor left	8B	minus
93	N	63	Cursor right	8D	Punkt
CF	O	5F	F1	8F	komma
CD	P	5D	F2	A9	doppel Kreuz
DF	Q	5B	F3	C9	plus
D9	R	59	F4	CB	
BD	S	57	F5	AD	
D7	T	55	F6	AB	-
D3	U	53	F7	E9	ß
97	V	51	F8	E5	\
DD	W	4F	F9	9F	<
9B	X	4D	F10	77/79	Return/Enter
9D	Y	7F	Space	41	Help
D5	Z	75	ESC		

Wenn man vom Tasten-Code eins subtrahiert, erhält man den Code für die losgelassene Taste. Das Beispiel Programm zeigt, wie man das "A" abfragt:

```
Loop: cmp.b #$BF,$BFEC01     ; Wurde Taste A gedrückt ?
      bne Loop               ; wenn nicht, dann weiter abfragen
```

7. Kapitel

Simple Sprites

- Simple Sprites
- Aufbau von Sprites
- Attached-Sprites
- Simple-Sprite-Routinen

7. Simple Sprites

Der Amiga bietet zwei Möglichkeiten, Simple Sprites darzustellen. Entweder programmiert man sie direkt ber die Hardware, was jedoch viel zu kompliziert und aufwendig ist oder man benutzt die Spriteroutinen, die die graphics.library des Betriebsystems zur Verfügung stellt. Mit Hilfe dieser Routinen ist es wirklich "simple" ein Sprite auf dem Bildschirm darzustellen und zu manipulieren. Doch bevor wir auf die Programmierung der Routinen eingehen, erst noch ein paar allgemeine Informationen zum Aufbau der selben.

7.1 Aufbau von Sprites

Es können maximal acht Sprites dargestellt werden, denen eine Nummer von Null bis Sieben zugeteilt wird, jedoch Sprite Null schon für den Mauszeiger reserviert ist.

Die Breite eines Sprites darf höhstens 16 Punkte betragen, aber die Höhe hingegen ist beliebig wählbar.

Sprites werden immer in der kleinsten Auflösung dargestellt, also 320 X 256 Pixel Lo-Res. Sie können aber in jedem Modus benutzt werden. Koordinaten außerhalb diesen Bereiches veranlassen das Sprite den sichtbaren Bildschirmbereich zu verlassen.

Ein Sprite kann maximal in vier Farben dargestellt werden, wobei sich jedoch immer zwei Spritepaare vier Farbregister teilen müssen. Von den 32 Farbregistern des Amiga werden die Register 16 bis 31 für die Sprites verwendet (siehe Tabelle).

Die Form eines Sprites wird Zeilenweise beschrieben, wobei jede Zeile durch zwei Worte (je 16 Bit) dargestellt wird. Jedes Bit des ersten Wortes (High-Word) bestimmt mit dem entsprechenden Bit des zweiten Wortes (Low-Word) die Farbe eines Punktes, also das Farbregister, auf welches zugegriffen wird.

Hier ein Beispiel zur Beschreibung einer Spritezeile:

High-Word	Low-Word
1100000000000000	1010000000000000

Punkt 1	: Bitpaar = 11
Punkt 2	: Bitpaar = 10
Punkt 3	: Bitpaar = 01
Punkt 4 - 16	: Bitpaar = 00

Wir sind wie oben beschrieben vorgegangen. Das erste Bit des ersten Wortes zusammen mit dem ersten Bit des zweiten Wortes ergibt für Punkt 1 das Bitpaar 11. Und so geht man alle 16 Punkte einer Zeile durch. Nach dem 16 Bit des zweiten Wortes (Low-Word) folgt dann das erste Word (High-Word) der nächsten Zeile wo sich dann die Farbe wie eben beschrieben wieder errechnet.

An diesem Beispiel wird auch ersichtlich, warum nur vier Farben zur Verfügung stehen. Mit zwei Bits (High and Low) lassen sich halt nur Werte von Null bis Drei darstellen.

Die folgende Tabelle gibt aufschluß darüber, welche Farbregister, bei welchen Bitpaaren für die Farben der jeweiligen Spritenummern herangezogen werden:

Sprite-Nr.	Bitpaar	Farbregisternummer
0 und 1	00	16
	01	17
	10	18
	11	19

2 und 3	00	20
	01	21
	10	22
	11	23
4 und 5	00	24
	01	25
	10	26
	11	27
6 und 7	00	28
	01	29
	10	30
	11	31

Die beiden Words der ersten Spritezeile werden nicht zur Farbermittlung herangezogen, sondern werden als Control-Words bezeichnet. Die beiden Control-Words enthalten Höhe und Position des Sprites. Um diese Control-Words brauchen wir uns allerdings nicht zu kümmern, dies erledigen später die Sprite-Routinen des Systems für uns.

7.2 Attached-Sprites

Wenn man im zweiten Sprite-Control-Word das Bit 7 auf Eins setzt, dann werden zwei Sprites zu einem Attached-Sprite zusammengefaßt. Hierbei können folgende Sprites kombiniert werden:

Sprite 1 mit Sprite 0
Sprite 3 mit Sprite 2
Sprite 5 mit Sprite 4
Sprite 7 mit Sprite 6

Die Sprites müssen allerdings die gleiche Größe, Form und Position haben.

Der Vorteil von einem Attached-Sprite ist, daß nunmehr 16 Farben für ein Sprite zur Verfügung stehen (Farbregister 16 bis 31). Jedoch sind auch diese Sprites weiterhin nur 16 Pixel breit.

Durch Kombination der Spritezeile des einen Sprites mit der Spritezeile des anderen Sprites, erhält man für jeden Punkt eine 4-Bitzahl, also einen Wert von 0 bis 15, die die Farbregister 16 bis 31 anwählt.

Hier ein Beispiel zur Beschreibung einer Attached-Spritezeile:
(Sprite 1 kombiniert mit Sprite 0)

Sprite1-HighWord : 0000000000000000
Sprite1-LowWord : 1000000000000000
Sprite0-HighWord : 1000000000000000
Sprite0-LowWord : 1000000000000000

Punkt 1 : Bitkombination = 0111
Punkt 2 - 16 : Bitkombination = 0000

Die Bitpaare werden hier genauso errechnet wie bei einfachen Sprites, mit dem Unterschied daß man hier vier Words für eine Spritezeile hat, statt zwei.

Folgende Farbregister werden somit zugewiesen:

Bitkombination	**Farbregisternummer**
0000	16
0001	17
0010	18
0011	19
0100	20

0101	21
0110	22
0111	23
1000	24
1001	25
1010	26
1011	27
1100	28
1101	29
1110	30
1111	31

7.3 Simple-Sprite-Routinen

Bevor wir nun ein Simple-Sprite auf dem Bildschirm erscheinen lassen können, müssen wir zunächst uns einen vom System zuweisen lassen. Für diesen Zweck stellt uns die graphics.library die Routine "GetSprite ()" zur Verfügung. Dieser übergeben wir die Spritenummer des gewünschten Sprites (von 0 bis 7), dabei nicht vergessen, das Sprite 0 für den Mauspointer reserviert ist. Außerdem wird in A0 noch die Anfangsadresse einer SimpleSprite-Struktur verlangt. Denn woher soll das System wissen, ab welcher Adresse die Spritedaten beginnen, wie Breit es ist oder welche Position es besitzt. Diese Informationen sind alle in der SimpleSprite-Struktur enthalten. Dabei zeigt das erste LongWord auf die Spritedaten, welche mit den beiden ControllWords beginnen und am Ende der Spritedaten mit zwei Null-Words enden.

```
dc.l spritedatas        ; Zeiger auf Spritedaten
```

Danach folgt ein Word, das die Höhe des Sprites enthält.

```
dc.w spritehöhe
```

Die nächsten zwei Words enthalten die Position des Sprites. Zuerst die horizontale (X-Richtung), dann die vertikale (Y-Richtung) Position.

```
dc.w X-Position
dc.w Y-Position
```

Das letzte Word enthält die Spritenummer, welche diesem Sprite zugeteilt wurde.

```
dc.w spritenummer
```

Hier die ganze SimpleSprite-Struktur nochmal im Zusammenhang:

SimpleSprite-Struktur: (Länge = 12 Bytes)

Offset	Typ	Bezeichnung	Beschreibung
000	Long	Datas	Zeiger auf Spritedaten
004	Word	Höhe	Höhe des Sprites
006	Word	X	X-Position des Sprites
008	Word	Y	Y-Position des Sprites
010	Word	Nummer	Nummer des Sprites (0 bis 7)
012		END	Ende der SimpleSprite-Struktur

Hier noch ein Beispiel für eine vollständige SimpleSprite-Struktur:

```
dc.l spritedatas        ; Zeiger auf Spritedaten
dc.w 3                  ; Höhe = 3 Zeilen
dc.w 100                ; X-Position = 100
dc.w 60                 ; Y-Position = 60
dc.w 1                  ; Spritenummer = 1
```

```
spritedatas:
 dc.w 0,0              ; beiden ControlWords
 dc.w $ffff,$ffff      ; erste Zeile
 dc.w $0013,$2341      ; zweite Zeile
 dc.w $1234,$fff0      ; dritte Zeile
 dc.w 0,0              ; End of Datas
```

Nachdem wir die Routine "GetSprite ()" aufgerufen haben, erhalten wir in D0 die Spritenummer zurück, welche wir vorher übergeben haben. Konnte jedoch der Sprite nicht zugewiesen werden, wird in D0 eine -1 zurückgegeben. Falls dies eintritt, sollten sie diesen Sprite erst wieder freigeben, bevor sie sich diesen mit 'GetSprite ()' zuweisen lassen.

Das Freigeben von Sprites erledigt die Routine "FreeSprite ()" für uns. Ihr brauchen wir nur die gewünschte Spritenummer zu übergeben und der Sprite wird zur weiteren Verwendung freigegeben.

Ein kleiner Tip:

Bevor sie sich einen Sprite mit "GetSprite ()" zuweisen lassen, sollten sie immer diesen mit "FreeSprite ()" freigeben.

Jetzt haben wir uns zwar einen Sprite zuweisen lassen, jedoch erscheint dieser noch nicht auf dem Bildschirm. Denn bevor dies geschehen kann, müssen die Parameter, welche in der SimpleSprite-Struktur enthalten sind, in die entsprechenden Hardware-Register kopiert werden. Diese Register wiederum, sind in der Copperliste enthalten, damit sie ständig mit den selben Werten geladen werden. Die Adresse der Copperliste finden wir in der ViewPort-Struktur. Die Routine "ChangeSprite ()" erledigt all diese Aufgaben und läßt den Sprite auf dem Bildschirm erscheinen. Dazu benötigt sie die SimpleSprite-Struktur des gewünschten Sprites, die Adresse des ViewPorts und die Adresse der Spritedatas.

Haben sie vor Sprites nicht nur darzustellen sondern auch zu bewegen?

Dazu laden sie die Datenregister D0 und D1 mit den gewünschten Positionen. Da dies allcine natürlich nicht genügt, denn die Copperliste muß noch mit den neuen Werten geladen werden, benötigen wir wieder die ViewPort-Adresse. Damit das System weiß, welcher Sprite bewegt werden soll, wird noch die Adresse der SimpleSprite-Struktur vom entsprechenden Sprite übergeben. Nun können wir dic Routinc "MovcSpritc ()" aufrufcn, dic unscren Sprite an den neuen Koordinaten erscheincn läßt.

Es folgt eine kurze Beschreibung der eben beschriebenen vier Sprite-Routinen:

```
Routine   : FreeSprite (Nummer) (D0)
Library   : graphics.library
Offset    : -414 = -$19e
Parameter : D0 = Spritenummer (von 0 bis 7)
Rückgabe  : keine
Erklärung : Diese Routine gibt ein reserviertes Sprite vom System
            wieder frei.
-------------------------------------------------------------------

Routine   : GetSprite (SimpleSprite,Nummer) (A0,D0)
Library   : graphics.library
Offset    : -408 = -$198
Parameter : A0 = Zeiger auf SimpleSprite-Struktur des Sprites
            D0 = Spritenummer (von 0 bis 7)
Rückgabe  : D0 = Spritenummer welcher vorher übergeben wurde
               = -1 für Sprite konnte nicht reserviert werden
Erklärung : Diese Routine reserviert einen Sprite, welcher dann ma-
            nipuliert werden kann.
-------------------------------------------------------------------
```

```
Routine   : ChangeSprite (ViewPort,SimpleSprite,Data) (A0,A1,A2)
Library   : graphics.library
Offset    : -420 = -$1a4
Parameter : A0 =  Zeiger auf Viewport in dem der Sprite dargestellt
                  werden soll.
            A1 =  Zeiger auf SimpleSprite-Struktur des Sprites
            A2 =  Zeiger auf Spritedatas (Adresse steht in der Sim-
                  ple-Sprite-Struktur)
Rückgabe  : keine
Erklärung : Diese Routine aktiviert einen Sprite auf dem Bildschirm.
-----------------------------------------------------------------

Routine   : MoveSprite (ViewPort,SimpleSprite,X,Y) (A0,A1,D0,D1)
Library   : graphics.library
Offset    : -426 = -$1aa
Parameter : A0 =  Zeiger auf Viewport in dem der Sprite dargestellt
                  werden soll.
            A1 =  Zeiger auf SimpleSprite-Struktur des Sprites
            D0 =  X-Position an der der Sprite erscheinen soll
            D1 =  Y-Position an der der Sprite erscheinen soll
Rückgabe  : keine
Erklärung : Diese Routine laßt den Sprite auf die angegebenen Koor-
            dinaten erscheinen.
```

Wenn man eine ViewPort-Struktur benutzt, muß im eigenem Screen das "Sprite-Flag" in "ViewModes" bei der Screen-Struktur gesetzt werden.

Zum Schluß des Kapitels noch ein Beispielprogramm, was Simple-Sprites erzeugt und animiert!

```
;-------------------- Programmname = SimpleSprite ------------------
;-------------- Animiert ein SimpleSprite und wechselt -------------
;----------------------- dabei das Aussehen ------------------------
;------------------ wartet auf rechte Maustaste --------------------

 execbase = 4
 openlibrary = -552
 closelibrary = -414
;
 getsprite = -408
 freesprite = -414
 changesprite = -420
 movesprite = -426
;

start:
;---------- graphics.library öffnen ----------

 move.l execbase,a6            ; Execbasisadresse in a6 laden
 move.l #gfxname,a1            ; Zeiger auf Librarynamen
 clr.l d0                      ; Version 0
 jsr openlibrary(a6)           ; Library öffnen
 move.l d0,gfxbase             ; Grafik-Basisadresse in gfxbase
                               ; speichern

;---------- Sprite 1 freigeben ----------

 move.l #1,d0                  ; Spritenummer = 1
 move.l gfxbase,a6             ; Grafikbasisadresse in a6 laden
 jsr freesprite(a6)            ; Sprite 1 freigeben

;---------- Sprite 1 reservieren ----------

 move.l #1,d0                  ; Spritenummer = 1
 move.l #simplesprite,a0       ; Zeiger auf SimpleSprite-Struk-
                               ; tur in a0
```

```
 move.l gfxbase,a6              ; Grafikbasisadresse in a6 laden
 jsr getsprite(a6)              ; Sprite 1 reservieren

;---------- Sprite 1 auf Bildschirm aktivieren ----------

 move.l #0,a0                   ; keine bestimmte Viewport-Struktur
 move.l #simplesprite,a1        ; Zeiger auf SimpleSprite-Struktur in
                                ; a1
 move.l #spritedatas,a2         ; Zeiger auf Spritedatas in a2
 move.l gfxbase,a6              ; Grafikbasisadresse in a6 laden
 jsr changesprite(a6)           ; Sprite 1 auf Screen aktivieren

;-------- Auf rechte Mousetaste in Port 1 warten und Sprite animieren

main:
 btst #10,$dff016               ; rechte Maustaste
 beq ende                       ; gedrückt ?
 jsr animat                     ; wenn nicht, dann sprite 1 animieren
 bra main                       ; das ganze wieder von vorne

;---------- Sprite 1 animieren und Image(Aussehen) wechseln ---------

animat:
 add.l #1,starten               ; Speicherstelle(.l) um eins erhöhen
 move.l speed,d0                ; Spritegeschwindigkeit in D0 laden
 cmp.l starten,d0               ; Wenn starten = speed dann
 beq animat1                    ; muß Sprite 1 bewegt werden
 rts                            ; sonst keine Bewegung

animat1:
 clr.l starten                  ; Speicherstelle starten wieder auf
                                ; Null setzen
 cmp.w #1,richtung              ; Bewegt sich sprite 1 nach rechts?
 bne links                      ; oder nach links?
 cmp.w #304,xpos                ; Sprite 1 schon bei x-position 304
 bne nein                       ; Nein ?
```

```
 sub.w #1,xpos                  ; Sprite 1 um einen Punkt nach links
 move.w #0,richtung             ; Sprite 1 muß sich jetzt nach links
                                ; bewegen
 move.l #newspritedatas,a2      ; Zeiger auf neues Sprite-Image
 jsr changeimage                ; von Sprite 1 das Image wechseln
 jsr bewegen                    ; Sprite 1 auf neue Koordinaten setzen
 rts

nein:
 add.w #1,xpos                  ; Sprite 1 um einen Punkt nach rechts
 jsr bewegen                    ; Sprite 1 auf neue Koordinaten setzen
 rts

links:
 cmp.w #0,xpos                  ; Sprite eins schon bei x-position 0
 bne nein1                      ; Nein ?
 add.w #1,xpos                  ; Sprite 1 um einen Punkt nach rechts
 move.w #1,richtung             ; Sprite 1 muß sich jetzt nach rechts
                                ; bewegen
 move.l #spritedatas,a2         ; Zeiger auf altes Sprite-Image
 jsr changeimage                ; von Sprite 1 das Image wechseln
 jsr bewegen                    ; Sprite 1 auf neue Koordinaten setzen
 rts

nein1:
 sub.w #1,xpos                  ; Sprite 1 um einen Punkt nach links
 jsr bewegen                    ; Sprite 1 auf neue Koordinaten setzen
 rts

;---------- Sprite 1 auf neue Koordinaten setzen ----------

bewegen:
 move.l #0,a0                   ; Keine eigene Viewport-Struktur
 move.l #simplesprite,a1        ; Zeiger auf SimpleSprite-Struktur
 move.w xpos,d0                 ; neue x-position in D0 laden
 move.w #125,d1                 ; y-position = 125 in D1 laden
```

```
 move.l gfxbase,a6             ; Grafikbasisadresse in a6 laden
 jsr movesprite(a6)            ; Sprite auf neue Koordinaten setzen
 rts

;---------- Image von Sprite 1 wechseln ----------

changeimage:
 move.l #0,a0                  ; keine eigene Viewport-Struktur
 move.l #simplesprite,a1       ; Zeiger auf SimpleSprite-Struktur von
                               ; Sprite 1
 move.l gfxbase,a6             ; Grafikbasisadresse in a6 laden
 jsr changesprite(a6)          ; Sprite Image wechseln
 rts

;---------- Sprite 1 wieder freigeben ----------

ende:
 move.l #1,d0                  ; Spritenummer = 1
 move.l gfxbase,a6             ; Grafikbasisadresse in a6 laden
 jsr freesprite(a6)            ; Sprite 1 wieder freigeben

;---------- graphics.library schließen ----------

 move.l gfxbase,a1             ; Grafikbasisadresse in a1 laden
 move.l execbase,a6            ; Execbasisadresse in a6
 jsr closelibrary(a6)          ; Grafiklibrary schließen
 rts                           ; Ende

;---------- Parameter ----------

gfxname: dc.b "graphics.library",0
 even
gfxbase: dc.l 0
speed: dc.l 300                ; Bewegungsgeschwindigkeit von Sprite 1
richtung: dc.w 0               ; Sprite 1-Richtung (0 = links, 1 =
                               ; rechts)
```

```
xpos: dc.w 0                    ; x-position fⁿr Sprite 1
starten: dc.l 0                 ; Vergleichswert für Speed

simplesprite:
 dc.l spritedatas               ; Zeiger auf Spritedatas
 dc.w 6                         ; höhe = 6 punkte
 dc.w 160,125                   ; x- und y-position
 dc.w 1                         ; spritenummer = 1
;
spritedatas:
 dc.w 0.0                       ; SpriteControlwords auf Null setzen
 dc.w %0000000001100110,%0000000000100100  ;HIGH + LOWword zeile 1
 dc.w %0000000011001100,%0000000001001000  ;HIGH + LOWword zeile 2
 dc.w %0001000110011000,%0000100010010000  ;...
 dc.w %1100101100110000,%1100100100100000  ;...
 dc.w %0110011011000000,%0110001010000000  ;...
 dc.w %0011110110000000,%0011110100000000  ;HIGH + LOWword zeile 6
 dc.w 0,0                       ;End of Datas
;
newspritedatas:
 dc.w 0,0                       ;SpriteControlwords auf Null setzen
 dc.w %0001111000111111,%0001100000110101  ;HIGH + LOWword zeile 1
 dc.w %0011001100111111,%0010000000011010  ;HIGH + LOWword zeile 2
 dc.w %0011100000001100,%0001000000001100  ;...
 dc.w %0000111000001100,%0000101000001100  ;...
 dc.w %0110001100001100,%0110001000001100  ;...
 dc.w %0011111000001100,%0011111000001000  ;HIGH + LOWword zeile 6
 dc.w 0,0                       ;End of Datas

;-------------------------- Ende des Listings --------------------
```

8. Kapitel

Das Animationssystem

- Das Animationssystem
- VSprites
- VSprite-Struktur im Detail
- BOBs (BlitterObjekte)
- Bobroutinen des Systems
- Double-Buffer-BOBs
- BOB-Animation
- Prioritäten von Grafikobjekten
- Collisionsabfrage
- Tips & Tricks zu Grafikobjekten

8. Das Animationssystem

Jetzt endlich kommen wir wohl auf den Abschnitt des Buches, auf den Sie bestimmt schon lange gewartet haben. Denn was wäre der Amiga ohne seine bewegten Grafiken, wie sie so toll in Spielen anzusehen sind.

Doch bevor wir ins "Eingemachte" gehen, noch ein paar allgemeine Informationen zum Grafiksystem des Amiga.

Das Animationssystem, auch GEL-System genannt, verwaltet VSprites (V für Virtual = Schein), BOBs (Blitter Objekte) und AnimBOBs (Animierte Blitter Objekte). Das GEL-System steuert also die gesammten Grafikoperationen.

Das der Amiga so schnell Grafiken Darstellen und Bewegen kann, liegt daran, daß es dafür speziell einen Chip gibt. Dieser Chip nennt sich Blitter. Mit ihm ist es möglich unwahrscheinlich schnellen Datentransfer vorzunehmen. Er ist in der Lage eine Fläche von 16 Million Punkten in einer Sekunde zu füllen. Zusätzlich kann der Blitter auch noch Linien ziehen. Das Betriebsystem verwendet den Blitter beinahe für alle Grafikoperationen. Wie zum Beispiel die Textausgabe, Verschiebung von Fenstern, Darstellung von Menüs etc.

Allerdings ist die Programmierung des Blitters ziemlich schwer. Deswegen stellt uns das System entsprechende Routinen zur Verfügung, mit denen es ein leichtes ist, Grafiken erscheinen zu lassen.

8.1 VSprites

VSprites sind Objekte, die maximal 16 Punkte breit aber beliebig hoch sein können. Entsprechen dem Aufbau von Hardware-Sprites. Jedoch können von den VSprites, im Gegensatz zu den Hardware-Sprites, beliebig viele dargestellt werden.

Sie werden sich jetzt bestimmt fragen, wie es möglich ist, mehr als acht Sprites darzustellen. Denn schließlich liefert die Hardware nur für acht Sprites Register. Der Trick dabei ist, das mit Hilfe des Coppers die Spriteregister eines Sprites an einer späteren Rasterstrahlposition wieder mit neuen Werten geladen werden. Deswegen muß zwischen jedem VSprite mit gleicher Nummer mindestens eine Rasterstrahlposition Freiraum sein. Somit sind wieder nur maximal 8 VSprites nebeneinander möglich.

Bevor wir jedoch ein VSprite darstellen können, muß erstmal das GEL-System initialisiert sein. Dazu müssen wir eine GELsInfo-Struktur anlegen und mit der Routine "InitGels ()" mit den entsprechenden Startwerten füllen.

Hier nun eine Beschreibung der GELsInfo-Struktur und der Routine "InitGELs ()":

'GelsInfo'-Struktur: (Länge = 38 Bytes)

'InitGels ()' füllt diese Struktur mit den wichtigsten Werten!

Offset	**Typ**	**Bezeichnung**	**Beschreibung**
000	Byte	SprRsrvd	welche Sprites als VSprites genutzt werden
001	Byte	Flags	System-Flag
002	Long	GelHead	Zeiger auf erste 'VSprite'-Struktur

006	Long	GelTail	Zeiger auf letzte 'VSprite'-Struktur
010	Long	NextLine	Zeiger auf Puffer mit 8 Words (nur bei VSprites)
014	Long	LastColor	Zeiger auf Puffer mit 8 Longs (nur bei VSprites)
018	Long	CollHandler	Zeiger auf Puffer mit 16 Longs (Adressen der 16 verschiedenen Collisionsroutinen)
022	Word	Leftmost	linker Begrenzungsrand (für Collisions-Routine 0)
024	Word	Rightmost	rechter Begrenzungsrand (für Collisions-Routine 0)
026	Word	Topmost	oberer Begrenzungsrand (für Collisions-Routine 0)
028	Word	bottommost	unterer Begrenzungsrand (für Collisions-Routine 0)
030	Long	FirstBlissObj	wird nur vom System benutzt
034	Long	LastBlissObj	wird nur vom System benutzt
038		END	Ende der 'GelsInfo'-Struktur

'SprRsrvd', 'NextLine', 'LastColor', 'CollHandler' und der Begrenzungsrand müss°en, wenn gewünscht, selbst von einem eingetragen werden!

```
Routine  :  InitGels (GelHead,GelTail,GelsInfo) (A0,A1,A2)
Library  :  graphics.library
Offset   :  -120 = -$78
Parameter:  A0 =  Zeiger auf einen Puffer mit 58 Null-Bytes

            A1 =  Zeiger auf einen Puffer mit 58 Null-Bytes
                  A2 = Zeiger auf angelete `GelsInfo`-Struktur
Rückgabe :  keine
Erklärung:  Diese Routine initialisiert das Gelsystem. Die `GelsIn-
            fo`- Struktur braucht dann nur noch mit der `Rast-
            Port`-Struktur verbunden werden (Offset 20)!
```

Jetzt muß die GelsInfo-Struktur, wie oben schon erwähnt, in die RastPort- Struktur eingetragen werden, in welchem die ganzen VSprites erscheinen sollen. Dazu tragen Sie einfach die Anfangsadresse der GELsInfo-Struktur in die Rastportstruktur bei Offset 20 (GelsInfo) ein.

```
z. B. move.l #gelsinfo,rastport+20   ; wenn beide Strukturen selbst
                                       angelegt wurden
```

oder

```
      move.l rastport,a0             ; wenn RastPort-Struktur indirekt
      move.l #gelsinfo,20(a0)        ; erzeugt wurde
```

Welche Sprites für die VSprites zur Verfügung stehen, bestimmen Sie in der Variablen 'SprRsvrd' (GelsInfo-Struktur = Offset 0). Jedes Bit, welches hier gesetzt wird, kennzeichnet das VSprite, mit dazugehöriger Nummer, als reserviert. Dabei steht Bit 0 für VSprite 0 usw. Auch hier, wie bei den Simple-Sprites, ist zu beachten, daß das Sprite 0 für den Mauszeiger reserviert ist und deswegen muß Bit 0 immer auf Null stehen. Am besten setzen Sie 'SprRsvrd' auf den Wert $FE (= 254).

8.2 VSprite-Struktur im Detail

Doch wie initialisiert man die VSprites? Genau wie für die Simple-Sprites eine Struktur exestiert, exestiert auch für die VSprites eine Struktur. Sie hat den treffenden Namen 'VSprite'. In ihr sind alle nötigen Daten enthalten, die zur Darstellung eines VSprites nötig sind.

Die VSprite-Struktur wird aber nicht nur für die Darstellung von VSprites verwendet, sondern auch für die BOBs. Dazu jedoch später. Wir werden hier nur soweit auf die Beschreibung der VSprite-Struktur eingehen, wie es für die Programmierung, der selben, von nöten ist.

'VSprite'-Struktur: (Länge = 58 Bytes)

Offset	Typ	Bezeichnung	Beschreibung
000	Long	NextVSprite	Zeiger auf nächste 'VSprite'-Struktur
004	Long	PrevVSprite	Zeiger auf Vorgänger 'VSprite'-Struktur
008	Long	DrawPath	wird nur vom System benutzt
012	Long	ClearPath	wird nur vom System benutzt
016	Word	OldY	alte y-Position vom VSprite (System-Variable)
018	Word	OldX	alte x-Position vom VSprite (System-Variable)
020	Word	Flags	beschreibt wie das GEL zu behandeln ist:
	VSprite	Bit 0 = 1,	dann Gel ist ein VSprite sonst BOB
	SaveBack	Bit 1 = 1,	Hintergrund wird gespeichert
	Overlay	Bit 2 = 1,	nur gesetzte Bits des Gels werden in die Grafik kopiert
	MustDraw	Bit 3 = 1,	für VSprites (Farben speichern)
	BackSaved	Bit 8 = 1,	Hintergrund wurde Gespeichert
022	Word	Y	Y-Position des VSprites
024	Word	X	X-Position des VSprites
026	Word	Heigth	Höhe des VSprites (Anzahl Zeilen)
028	Word	Width	Breite des VSprites (Anzahl in Words)
030	Word	Depth	Anzahl Bitplanes (Tiefe)

032	Word	MeMask	siehe Kapitel Collisionen
034	Word	HitMask	siehe Kapitel Collisionen
036	Long	ImageData	Zeiger auf Grafikdaten vom VSprite (Datas)
040	Long	Borderline	siehe Kapitel Collisionen
044	Long	CollMask	siehe Kapitel Collisionen
048	Long	SprColors	Zeiger auf Puffer mit 4 Words für Farbspeicherung
052	Long	Bob	Zeiger auf 'BOB'-Struktur, (bei VSprites = 0)
056	Byte	PlanePick	unwichtig (auf Null setzen)
057	Byte	PlaneOnOff	unwichtig (auf Null setzen)
058		END	Ende der 'VSprite'-Struktur

Ausführliche Dokumentation der VSprite-Struktur:

Next/Prev-VSprite (Offset 000 und 004)

Diese beide Adressen zeigen auf die folgende und vorangegangene VSprite-Struktur. Diese werden automatisch vom System eingetragen. Sie brauchen sich also nicht darum zu kümmern.

DrawPath/ClearPath (Offset 008 und 012)

Werden für das ordnungsgemäße Darstellen von BOBs benutzt. Diese Parameter werden auch automatisch vom System verwaltet.

OldY/OldX (Offset 016 018)

In diesen Variablen werden die alten Positionen der BOBs gespeichert. Damit der Hintergrund wieder korrekt zurückgeschrieben werden kann. Wird auch vom System verwaltet und ist für die VSprite-programmierung nicht weiter von Bedeutung.

Flags (Offset 020)

Dieses Flag beschreibt, ob es sich bei der VSprite-Struktur um einen BOB oder einen VSprite handelt. Da wir ja VSprites benutzen wollen, muß hier Bit 0 auf eins gesetzt werden. (Flag = 1) Alle anderen Bits haben nur fr die Darstellung von BOBs eine Bedeutung.

Y/X (Offset 022 und 024)

Hier werden die Koordinaten des VSprites eingetragen, an denen es dann auf dem Bildschirm erscheinen soll. Erst die vertikale dann die horizontale Position.

Heigth und Width (Offset 026 und 028)

Hier steht die Höhe und Breite des VSprites. Die Höhe kann wie bei den Simple-Sprites beliebig gewählt werden. Da Sprites allerdings nur maximal 16 Punkte breit sein können, muß in Width eine 1 eingetragen werden. Eine Eins, weil die Breite in Words angegeben werden muß. Und ein Word ist ja 16 Bit groß. Wenn Sie hier keine Eins eingeben, kann es passieren, daß Ihr Sprite später nicht auf dem Bildschirm erscheint.

Depth (Offset 030)

Da Sprites nur 16 Punkte breit sind und sich nur aus einen HIGH und einem LOW-Word zusammensetzen, muß hier eine 2 eingetragen werden. Nur bei BOBs können hier bis zu 6 Planes eingetragen werden.

ImageData (Offset 036)

In diesen Long-Word wird die Anfangsadresse der SpriteDatas abgelegt, welche mit den beiden Controll-Words beginnen und mit den eigentlichen Grafikdaten fortgeführt werden. (Aufbau siehe Simple-Sprites)

SprColors (Offset 048)

Jedes VSprite kann seine eigenen Farben besitzen. Jedoch will man für ein Spritepaar (z. B. Nummer 0 und 1) verschiedene Farben benutzen, ist der Amiga nicht in der Lage dieses zu bewerkstelligen und ein Sprite würde ausfallen. Dieses können Sie vermeiden, indem Sie allen Sprites die gleichen Farben zuordnen. Und in der GELsInfo-Struktur den 'LastColor'-Zeiger auf einen Puffer von 32 Bytes zeigen lassen. Wenn jedoch VSprites sich außerhalb einer vertikalen Position von 200 befinden, nehmen Sie die Farbe des Mauspointers an. Dieses ist leider ein Fehler im Betriebsystem und läßt sich nicht umgehen.

Hier ein Beispiel einer korekt initialisierten VSprite-Struktur:

```
vsprite:
dc.l 0,0            ; Next/PrevVSprite
dc.l 0,0            ; Draw/ClearPath
dc.w 0,0            ; OldY/X
dc.w 1              ; Flag: GEL = VSprite
dc.w 40,50          ; Y = 40, X = 50 (Position)
dc.w 2              ; Höhe = 2 Zeilen
dc.w 1              ; Breite = 1 Word  (Festwert)
dc.w 2              ; Tiefe = 2 (Festwert)
dc.w 0,0            ; MeMask/HitMask
dc.l data           ; Imagedaten
dc.l 0,0            ; BorderLine/CollMask
dc.l 0              ; SprColors = keine eigenen Farben
dc.l 0              ; Keine BOB-Struktur
dc.b 0.0            ; PlanePick/OnOff

data:
dc.w 0,0            ; ControllWords
dc.w $100,$234      ; 1. Zeile HIGH und LOW-Word
dc.w $232,$232      ; 2. Zeile
dc.w 0,0            ; End of Datas
```

Es ist zu beachten, daß bei VSprites keine "Attached"-Sprites, wie bei den Simple-Sprites, erzeugt werden können.

Haben Sie nun die VSprite-Struktur soweit initialisiert, müssen Sie diese mit der Routine "AddVSprite ()" in dem vorher dem RastPort mit "InitGels ()" zugewiesenen GELsInfo-Struktur einfügen.

Bevor Sie jedoch die VSprites darstellen können, müssen Sie mit der Routine "SortGList ()" die GEL-Liste sortieren. Damit der Copper entlastet wird und die größtmögliche Anzahl von VSprites

korekt dargestellt werden kann.

Jetzt können Sie mit der Routine "DrawGList ()" eine neue Copperliste vom System generieren lassen. Und mit "RemakeDisplay ()" (bei Screens) oder mit "MakeVPort ()", "MrgCop ()" und "LoadView ()" diese neue Copperliste dann ausführen. Jetzt endlich erscheint Ihr VSprite auf dem Bildschirm.

Wenn Sie das Aussehen, die Position oder dergleichen vom VSprite ändern wollen, ändern Sie einfach die entsprechenden Zeiger und sortieren danach mit "SortGList ()" die GEL-Liste wieder neu. Danach generieren Sie wieder eine neue Copperliste (DrawGList) und führen wieder "RemakeDisplay ()" aus.

Die Beschreibung der eben erwähnten Routinen darf hier natürlich auch nicht fehlen:

Routine : AddVSprite (VSprite,RastPort) (A0,A1)

```
Library  : graphics.library
Offset   : -102 = -$66
Parameter: A0 =  Zeiger auf angelegte `VSprite`-Struktur
           A1 =  Zeiger auf `RastPort`-Struktur
Rückgabe : keine
Erklärung: Diese Routine führt ein VSprite zum Gelsystem hinzu. Um
           ein VSprite erscheinen zu lassen, m"ssen die Copperlisten
           neu berechnet werden!
```

Routine : RemVSprite (VSprite) (A0)

```
Library  : graphics.library
Offset   : -138 = -$8a
Parameter: A0 = Zeiger auf angelegte `VSprite`-Struktur
Rückgabe : keine
Erklärung: Diese Routine entfernt ein VSprite für immer aus dem
           Gelsystem!
```

```
Routine  :  SortGList (RastPort) (A1)
Library  :  graphics.library
Offset   :  -150 = -$96
Parameter:  A1 = Zeiger auf `RastPort`-Struktur
Rückgabe :  keine
Erklärung:  Diese Routine sortiert das Gelsystem (Position etc.). Muß
            vor jedem `DrawGList ()` aufgerufen werden!
```

```
Routine  :  DrawGList (RastPort,ViewPort) (A1,A0)
Library  :  graphics.library
Offset   :  -114 = -$72
Parameter:  A0 = Zeiger auf `ViewPort`-Struktur
            A1 = Zeiger auf `RastPort`-Struktur
Rückgabe :  keine
Erklärung:  Diese Routine generiert für alle VSprites eine neue Cop-
            perliste.
```

Das Beispielprogramm darf hier natürlich auch nicht fehlen. Es erzeugt einen Screen, auf dem ein VSprite bewegt wird. Wenn die rechte Maustaste betätigt wird, wird das Programm beendet.

```
;--------------------- Programmname = VSprites --------------------
;--------------------- rechte Maustaste = Ende --------------------

 move.l #intname,a1           ; Zeiger auf Intuitionname
 clr.l d0                     ; Version = 0
 move.l 4,a6                  ; Execbase nach A6
 jsr -552(a6)                 ; OpenLibrary
 move.l d0,intbase            ; Intuitionbasis speichern
 tst.l d0                     ; Error ?
 beq exit
```

```
move.l #gfxname,a1              ; Zeiger auf graphicsname
clr.l d0                        ; Version = 0
move.l 4,a6                     ; Execbase nach A6
jsr -552(a6)                    ; OpenLibrary
move.l d0,gfxbase               ; Graphicsbasis speichern
tst.l d0                        ; Error ?
beq exit

move.l #OSArgs,a0               ; Zeiger auf Screenargumentenli-
                                ; ste
move.l intbase,a6               ; Intuitionbasis nach A6
jsr -198(a6)                    ; OpenScreen
move.l d0,screen                ; Adresse der Screendatenstruktur
                                ; speichern
tst.l d0                        ; Error ?
beq exit

move.l screen,d0                      ; Screenadresse nach D0
add.l #44,d0                          ; ViewPort-Adresse ermitteln
move.l d0,viewport                    ; und zwischenspeichern
move.l screen,d0                      ; Screenadresse nach D0
add.l #84,d0                          ; RastPort-Adresse ermitteln
move.l d0,rastport                    ; und zwischenspeichern

lea gelhead,a0                  ; Zeiger auf erste VSprite-Struktur
lea geltail,a1                  ; Zeiger auf letzte VSprite-Struktur
lea gelsinfo,a2                 ; Zeiger auf GelsInfo-Struktur
move.l gfxbase,a6               ; graphics-basis nach A6
jsr -120(a6)                    ; InitGels
move.l rastport,a0              ; RastPort-Adresse nach A0
move.l #gelsinfo,20(a0)         ; GELsInfo-Struktur in RastPort ein-
                                ; hängen
lea vsprite,a0                  ; Zeiger auf VSprite-Struktur
move.l rastport,a1              ; Zeiger auf RastPort-Struktur
move.l gfxbase,a6
jsr -102(a6)                    ; AddVSprite ()
```

```
main:
 btst #10,$dff016                   ; rechte Maustaste gedrückt ?
 beq exit

 move.l rastport,a1                 ; RastPort-Struktur nach A1
 move.l gfxbase,a6
 jsr -150(a6)                       ; SortGList ()

 move.l rastport,a1                 ; Zeiger auf RastPort-Struktur
 move.l viewport,a0                 ; Zeiger auf ViewPort-Struktur
 move.l gfxbase,a6                  ; graphicsbasis
 jsr -114(a6)                       ; DrawGList ()

 move.l gfxbase,a6
 jsr -270(a6)                       ; WaitTOF - damit der Sprite
                                    ; nicht so flackert

 move.l intbase,a6                  ; intuitionbasis
 jsr -384(a6)                       ; RemakeDisplay ()

 bsr animat                         ; VSprite bewegen

 bra main


;---------- Positionen des VSprites ändern ----------

animat:
 lea vsprite,a0                     ; VSpriteadresse nach A0
 move.w 24(a0),d0                   ; X-pos.
 cmp.w #310,d0                      ; schon bei 310 ?
 bpl aneg                           ; wenn nicht, dann weiter
 cmp.w #0,d0                        ; oder bei 0 ?
 bpl noneg                          ; wenn nicht, dann weiter
```

```
aneg:
 neg.w pixelx                   ; Vorzeichen von X-Speed ändern

noneg:
 move.w pixelx.d0
 add.w d0,24(a0)                ; X-Speed zur X-Pos addieren
 move.w 22(a0),d0               ; Y-pos.
 cmp.w #200,d0                  ; schon bei 200 ?
 bpl aneg2                      ; wenn nicht, dann weiter
 cmp.w #0,d0                    ; oder bei 0 ?
 bpl noneg2                     ; wenn nicht, dann weiter

aneg2:
 neg.w pixely                   ; Vorzeichen von Y-Speed ändern

noneg2:
 move.w pixely,d0
 add.w d0,22(a0)                ; Y-Speed zur Y-Pos addieren
 rts                            ; Unterprogramm Ende

;---------- Programm beenden ----------

exit:
 move.l screen,a0               ; Screendatenstruktur
 cmp.l #0,a0                    ; liegt eine Adresse vor ?
 beq exit2
 move.l intbase,a6              ; Intuitionbasis
 jsr -66(a6)                    ; CloseScreen

exit2:
 move.l intbase,a1              ; Intuitionbasis
 cmp.l #0,a1                    ; liegt eine Adresse vor ?
 beq exit3
 move.l 4,a6                    ; Execbasis
 jsr -414(a6)                   ; CloseLibrary
 move.l gfxbase,a1              ; graphicsbasis
```

```
 cmp.l #0,a1                          ; liegt eine Adresse vor ?
 beq exit3
 move.l 4,a6                          ; Execbasis
 jsr -414(a6)                         ; CloseLibrary

exit3:
 rts                                  ; Programmende


;----------- Parameter -----------
pixelx: dc.w 1                        ; Speed in X-Richtung
pixely: dc.w 1                        ; Speed in Y-Richtung
;
intname: dc.b "intuition.library",0
 even
intbase: dc.l 0
gfxname: dc.b "graphics.library",0
 even
gfxbase: dc.l 0
rastport: dc.l 0
viewport: dc.l 0

OSArgs:                               ; ab hier Screenargumentenliste
 dc.w 0,0                             ; x und y-position
 dc.w 320,256                         ; breite und höhe
 dc.w 5                               ; anzahl der Bitplanes (=32 Far-
                                      ; ben)
 dc.b 1,3                             ; Textfarben
 dc.w 0                               ; ViewMode = Normal
 dc.w 15                              ; Screen mit allen Funktionen
 dc.l 0                               ; kein eigener Font
 dc.l title                           ; Zeiger auf Titletext
 dc.l 0                               ; keine Gadget
 dc.l 0                               ; keine eigene Bitmap-Struktur
title: dc.b "Press Right Mouse to Exit",0
 even
```

```
screen:
 dc.l 0                  ; hier wird Screendatenstruktur gespeichert


gelhead: blk.b 58,0      ; Zeiger auf erste VSprite-Struktur
geltail: blk.b 58,0      ; Zeiger auf letzte VSprite-Struktur

gelsinfo:                ; ab hier GelsInfo-Struktur
 dc.b 252                ;SprRsrvd: alle Sprites bis Mauspointer und 1
 dc.b 0                  ; System-Flag
 dc.l 0,0                ; GelHead,GelTail
 dc.l nextline
 dc.l lastcolor
 dc.l collhandler
 dc.w 0,0,0,0            ; Begrenzungsrand unwichtig
 dc.l 0,0                ; First/LastBlissObjekt

nextline: blk.w 8,0      ; Puffer von 8 Words
lastcolor: blk.l 8,0     ; Puffer von 8 Longs
collhandler: blk.l 16,0 ; Puffer von 16 Longs

vsprite:                 ; ab hier VSprite-Struktur
 dc.l 0,0                ; Next/PrevVSprite
 dc.l 0,0                ; Draw/ClearPath
 dc.w 0,0                ; OldY/X
 dc.w 1                  ; Flag: GEL = VSprite
 dc.w 50,70              ; Y,X-Position
 dc.w 6,1                ; Höhe = 6, Breite = 1 Word
 dc.w 2                  ; Tiefe = 2
 dc.w 0,0                ; MeMask/HitMask
 dc.l imagedata          ; Grafikdaten des Sprites
 dc.l 0                  ; BorderLine
 dc.l 0                  ; CollMask
 dc.l SprColors          ; SprColors
 dc.l 0                  ; keine BOB-Struktur
 dc.b 0,0                ; PlanePick/OnOff
```

```
SprColors: dc.w 1000,2000,3000,4000          ; Farben vom VSprite

imagedata:
 dc.w %0000000001100110,%0000000000100100   ;HIGH + LOWword zeile 1
 dc.w %0000000011001100,%0000000001001000   ;HIGH + LOWword zeile 2
 dc.w %0001000110011000,%0000100010010000   ;...
 dc.w %1100101100110000,%1100100100100000   ;...
 dc.w %0110011011000000,%0110001010000000   ;...
 dc.w %0011110110000000,%0011110100000000   ;HIGH + LOWword zeile 6
 dc.w 0,0

;-------------------------- Listingende ------------------------
```

8.3 BOBs (BlitterObjekte)

BOBs - die Abkürzung für BlitterObjekte - werden softwaremäßig erzeugt. Es können beliebig viele, in beliebiger Größe und in einer beliebigen Anzahl von Farben dargestellt werden.

Wie der Name schon sagt, werden BOBs mit Hilfe des Blitters erzeugt und manipuliert. Und da der Blitter für das Darstellen von Grafiken, das sogenannte Blitterfenster in Word-Breite benötigt, muß die Breite eines BOBs durch 16 teilbar sein.

Die BOBs werden direkt in den Speicher der Bitplanes eines RastPorts hineinkopiert. Deswegen hängt die Auflösung immer von den gerade eingestelltem Modus ab.

BOBs sind also Teile des eigentlichen Grafikbildes. Doch durch das Kopieren in das Grafikbild, würde der Hintergrund normalerweise Verloren gehen. Damit dies nicht geschieht, kann man durch setzen bestimmter Flags, dem Blitter mitteilen, den Hintergrund vorher zwischen zu speichern.

Das neuberechnen der Copperlisten entfällt bei den BOBs, da sie sofort dargestellt werden. Denn das Grafikbild wird ja automatisch durch den Copper ständig erneuert.

Genauso wie VSprites, sind auch die BOBs teile des GEL-Systems. Bevor wir also BOBs erzeugen können, muß erstmal das Gelsystem initialisiert werden. Wie bei den VSprites muß eine GelsInfo-Struktur angelegt, mit der Routine "InitGels ()" initialisiert und diese dann in die RastPort-Struktur eingebunden werden.

Jetzt brauchen wir nur noch eine VSprite-Struktur für den BOB anlegen und diese mit den entsprechenden Werten füllen. Es folgt der Aufbau und die Beschreibung der VSprite-Struktur, wie es für die Erzeugung von BObs erforderlich ist.

'VSprite'-Struktur: (Länge = 58 Bytes)

Offset	Typ	Bezeichnung	Beschreibung
000	Long	NextVSprite	Zeiger auf nächste 'VSprite'-Struktur
004	Long	PrevVSprite	Zeiger auf Vorgänger 'VSprite'-Struktur
008	Long	DrawPath	wird nur vom System benutzt
012	Long	ClearPath	wird nur vom System benutzt
016	Word	OldY	alte y-Position vom BOB (System-Variable)
018	Word	OldX	alte x-Position vom BOB (System-Variable)
020	Word	Flags	beschreibt wie das GEL zu behandeln ist:
	SaveBack	Bit 1 = 1,	Hintergrund wird gespeichert
	Overlay	Bit 2 = 1,	nur gesetzte Bits des BOBs werden in die Grafik kopiert
022	Word	Y	Y-Position des BOBs
024	Word	X	X-Position des BOBs

026	Word	Heigth	Höhe des BOBs (Anzahl Zeilen)
028	Word	Width	Breite des BOBs (Anzahl in Words)
030	Word	Depth	Anzahl Bitplanes (Tiefe)
032	Word	MeMask	siehe Kapitel Collisionen
034	Word	HitMask	siehe Kapitel Collisionen
036	Long	ImageData	Zeiger auf Grafikdaten vom BOB (Datas)
040	Long	Borderline	siehe Kapitel Collisionen (Wichtig)
044	Long	CollMask	siehe Kapitel Collisionen
048	Long	SprColors	unwichtig
052	Long	Bob	Zeiger auf 'BOB'-Struktur
056	Byte	PlanePick	siehe unten
057	Byte	PlaneOnOff	siehe unten
058		END	Ende der 'VSprite'-Struktur

Ausführliche Beschreibung der VSprite-Struktur:

Die Parameter mit den Offsets 000 bis 018 sind Systemvariablen, um die man sich nicht zu kümmern braucht.

Flags (Offset 020)

Wenn der Hintergrund von einen BOB zwischengespeichert werden soll, dann muß hier das Bit 1 gesetzt werden. Ansonsten geht der Hintergrund verloren. Wird das Overlay-Bit gesetzt, werden nur die gesetzen Punkte eines BOBs in den RastPort kopiert. übeall wo eine Null steht, scheint der Hintergrund durch. Der BOB wird sozusagen transparent über das Bild gelegt.

Y/X (Offset 022 und 024)

Hier befinden sich die Koordinaten des BOBs. Erst die vertikale und danach die waagerechte Position. BOBs können sich auch außerhalb des oder teilweise im RastPort befinden. Denn alle Bits die sich außerhalb der möglichen Koordinaten befinden, werden, vor der Ausgabe auf dem Screen, vom System abgeschnitten.

Heigth/Width (Offset 026 und 028)

Damit das System wei, wie hoch und breit ein BOB ist, müssen Sie hier die entsprechenden Werte eintragen. Dabei wird die Höhe, wie gewohnt in Zeilen aber die Breite hingegen in Words angegeben. Nehmen wir mal an der BOB wäre 18 Punkte (Pixel) breit, dann müßte sie bei "Width" eine zwei entragen.

Depth (Offset 030)

Da ein BOB aus beliebig vielen Farben bestehen kann und die Farbanzahl von der Anzahl Bitplanes abhängt, wird hier die Anzahl der verwendeten Bitplanes eingetragen. Dabei darf die Tiefe (Depth) des BOBs nicht die Tiefe vom Screen überschreiten, dieses könnte zum Abstürzen (Guru...) führen.

ImageData (Offset 036)

Hier steht die Anfangsadresse der ersten Bitplane vom BOB. Die Bitplanes müssen hintereinander im Speicher liegen. Also erst Bitplane 1 dann 2 usw.

BOB (Offset 052)

Für einige Spezialeffekte braucht das System eine Zusatzstruktur, in der die gesuchten Informationen enthalten sind. Diese Struktur nennt sich "BOB" und die Anfangsadresse dieser wird hier eingetragen. Wie die Struktur aufgebaut ist, darauf wird unten weiter eingegangen.

PlanePick/PlaneOnOff (Offset 056 und 057)

PlanePick gibt an, welche Bitplanes des BOBs in die Grafik kopiert werden soll. Jedes Bit steht dabei für eine Bitplane (z. B. Bit 0 = Bitplane 0 etc.). Wenn der BOB z. B. aus 5 Bitplanes besteht, also in Depth eine 5 steht, müssen hier die ersten 5 Bits gesetzt werden (Bits 0 bis 4). Sonst wird keine Grafik dargestellt. PlaneOnOff gibt an, welche BitPlanes, die nicht angeschaltet sind, mit dem "ImageShadow" der BOB-Struktur beschrieben werden. Mit diesen beiden Variablen lassen sich verschiedene Farbkombinationen für den BOB herstellen.

Die BOB-Struktur wurde ja schon erwähnt, wie diese nun genau aufgebaut ist, darauf wird jetzt eingegangen:

'BOB'-Struktur: (Länge = 30 Bytes)

Muß mit 'VSprite'-Struktur verbunden werden (Offset 52)!

Offset	Typ	Bezeichnung	Beschreibung
000	Word	B_Flags	wie das BOB vom System behandelt werden soll:

```
User--SaveBOB     Bit 0 = 1, dann wirkt BOB wie ein Paintbrush
User--BOBisComp   Bit 1 = 1, Bob besitzt eine `AnimComp`-Struktur
```

BWaiting		Bit 8 = 1,	ein BOB auf den `Before` in der `BOB`-Struktur zeigt, wird zuerst gezeichnet
BDrawn		Bit 9 = 1,	BOB wurde gezeichnet
BobsAway		Bit10 = 1,	BOB wird beim nächsten `DrawGList ()` aus dem Gelsystem entfernt
BobNix		Bit11 = 1,	Bob wurde aus dem Gelsystem entfernt
002	Long	SaveBuffer	Zeiger auf Puffer zum speichern des Hintergrundes (nur wenn SaveBack-Flag in VSprite gesetzt ist) SaveBufferSize = bobhöhe * bobbreite * 2 * screentiefe
006	Long	ImageShadow	Zeiger auf Puffer in der die Maske aller gesetzten Bits von allen Bitmaps zusammen steht ImageShadowSize = bobhöhe * bobbreite * 2
010	Long	Before	Zeiger auf BOB-Struktur, die über diesen BOB gezeichnet werden soll (oder Null) - Priorität
014	Long	After	Zeiger auf BOB-Struktur, die hinter diesem BOB gezeichnet werden soll (oder Null) - Priorität
018	Long	VSprite	Zeiger auf dazugehörige VSprite-Struktur
022	Long	BobComp	Zeiger auf 'AnimComp'-Struktur, nur wenn Flag 'BOBisComp' in BOB-Struktur gesetzt (sonst Null)
026	Long	DBuffer	Zeiger auf 'DBufPakket'-Struktur (wenn BOB doppelt gepuffert werden soll - sonst Null)
030		END	Ende der 'BOB'-Struktur

Auch hier eine ausführliche Dokumentation:

B_Flags (Offset 000)

Mit diesem Flag werden die Charaktere des BOBs bestimmt. Wird das "SaveBOB"-Flag gesetzt, dann wird der Hintergrund nicht zwischengespeichert, sondern von BOB überschrieben. Dadurch ist es ziemlich leicht möglich den BOB als Brush-Pinsel, wie es z. B. DeluxePaint bietet, zu benutzen. Auf das "BOBisComp"-Flag wird im Animationskapitel eingegangen. Es muß gesetzt werden, wenn der BOB von System animiert werden soll. Die anderen Flags werden vom System genutzt.

SaveBuffer (Offset 002)

Hier muß die Adresse des Puffers eingetragen werden, in dem der Hintergrund zwischengespeichert wird. Natürlich nur, wenn das "SAVEBACK"-Flag in der dazugehörigen VSprite-Struktur gesetzt ist. Der Puffer muß von ihnen selbst reserviert werden. Die Größe des Puffers entsteht durch Multiplikation von Bobhöhe * Bobbreite * 2 * Screentiefe.

ImageShadow (Offset 006)

Dieser Puffer muß von ihnen reserviert werden. In dem Puffer wird, durch aufrufen der Routine "InitMasks ()", die Maske des BOBs abgelegt. Die Maske entsteht durch eine ODER-Verknüpfung aller vorhandenen Bitmaps des BOBs. Sie ist nötig zur richtigen Berechnung des BOBs, beim Kopieren in den Hintergrund. Die Puffergröße entspricht einer Bitmapgröße vom BOB.

Before/After (Offset 010 und 014)

Diese Adressen werden für die Berechnung der Prioritäten der BOBs benötigt. Wenn die Priorität egal sein soll, kann man diese Zeiger auf Null setzen. Wie diese Zeiger zu verwenden sind, wird im Kapitel Prioritäten beschrieben.

VSprite (Offset 018)

Hier muß die Adresse der dazugehörigen VSprite-Struktur von ihnen eingetragen werden, sonst erscheint kein BOB auf dem Bildschirm. Oder es kann sogar zum Abstürzen führen.

BobComp (Offset 022)

Dieser Zeiger wird für die automatische Animation eines BOBs durch das System benötigt. Er zeigt dann auf eine zusätzliche Struktur. Wenn sie den BOB allerdings selbst animieren wollen, brauchen sie hier einfach nur eine Null eintragen. Wie die Animation durch das System programmiert wird, steht im Kapitel "BOB-Animtion".

DBuffer (Offset 026)

Dieser Zeiger wird für das Programmieren von doppelt gepufferten BOBs benötigt. Durch das doppelt puffern, fällt das Flackern und Ruckeln des BOBs weg. Siehe Kapitel "Double-Buffer-BOBs".

Das war jetzt wieder recht viel trockender Stoff der abgehandelt wurde. Zum besseren Verständnis folgt nun ein Beispiel zur Initialisierung der VSprite- und BOB-Struktur eines BOBs.

```
vsprite:                              ; ab hier VSprite-Struktur
 dc.l 0,0                             ; Next/PrevVSprite
 dc.l 0,0                             ; Draw/ClearPath
 dc.w 0,0                             ; OldY/X
 dc.w 6                               ; Flag: SaveBack/Overlay
 dc.w 0,0                             ; Y,X-Position
 dc.w 3,2                             ; Höhe = 3, Breite = 2 Words
 dc.w 3                               ; Tiefe = 3 Bitmaps
 dc.w 0,0                             ; MeMask/HitMask
 dc.l imagedata                       ; Grafikdaten des BOBs
 dc.l borderline                      ; BorderLine (Size = Bobbreite)
 dc.l shadowmask                      ; CollMask = Shadowmask
 dc.l 0                               ; SprColors = keine
 dc.l bob                             ; Zeiger auf BOB-Struktur
 dc.b 7,0                             ; PlanePick = 7 für drei Bitmaps
                                      ; / OnOff = Null

imagedata:
 dc.w %1100000110000110,%0110111100000000   ; Zeile 1 v. Bitmap 1
 dc.w %1100011001100110,%0110110000000000   ; Zeile 2 v. Bitmap 1
 dc.w %1111011001100011,%1000110000000000   ; Zeile 3 v. Bitmap 1

 dc.w %0011110000000000,%0011110000000000   ; Zeile 1 v. Bitmap 2
 dc.w %0000000000000000,%0000000000000000   ; Zeile 2 v. Bitmap 2
 dc.w %0000011110000000,%0000001111000000   ; Zeile 3 v. Bitmap 2

 dc.w %0000000000011100,%0000000000000000   ; Zeile 1 v. Bitmap 3
 dc.w %0000011110000000,%0001111100000000   ; Zeile 2 v. Bitmap 3
 dc.w %0011100000000000,%0000000000000000   ; Zeile 3 v. Bitmap 3

bob:                        ; ab hier BOB-Struktur
 dc.w 0                     ; Flag = Null
 dc.l savebuffer            ; Zeiger auf Puffer für Hintergrundspeiche-
                            ; rung
 dc.l imageshadow           ; Zeiger auf Puffer wo Maske abgelegt wird
 dc.l 0,0                   ; Priorität egal
```

```
 dc.l vsprite              ; Zeiger auf dazugehörige VSprite-Struktur
 dc.l 0                    ; Keine BobComp-Struktur für Animation
 dc.l 0                    ; kein double-buffering
savebuffer: blk.b 3*2*2*5      ; Puffergröße
imageshadow: blk.b 3*2*2       ; Puffergröße
borderline: blk.b 2*2          ; Puffergröße
```

8.4 Bobroutinen des Systems

Um ein BOB in das Gel-System einhängen zu können, muß man als erstes die GelsInfo-, VSprite- und BOB-Struktur anlegen. Danach durch aufrufen von "InitGels ()" das Gelsystem initialisieren. Die GelsInfo-Struktur mit der RastPort-Struktur verbinden, in dem der BOB später erscheinen soll. Die Routine "InitMaks ()" setzt dann die ShadowMask vom BOB, in dem vorher reservierten Puffer. Jetzt endlich, kann man mit der Routine "AddBOB ()" den BOB zum Gelsystem hinzufügen.

Die Routine "SortGList ()" sortiert die Gelliste und "DrawGList ()" stellt die BOBs in der Gel-Liste auf der BitMap da.

Das ganze nochmal in Stichworten der Reihe nach sortiert:

1. 'GelsInfo'-, 'VSprite'- und 'BOB'-Struktur anlegen
2. 'InitGels ()' aufrufen ; Gelsystem initialisieren
3. 'GelsInfo'-Struktur mit 'RastPort'-Struktur verbinden (Offset 20)
4. 'InitMasks ()' aufrufen ; Masken initialisieren
5. 'AddBob ()' aufrufen BOB zum Gelsystem hinzufügen
6. 'SortGList ()' aufrufen ; Gelsystem sortieren
7. 'DrawGList ()' aufrufen ; BOB in die Bitmap zeichnen

Punkt 6 und 7 sind immer wieder durchzuführen, wenn ein BOB bewegt werden soll.

Wie die einzelnen Routinen aufgerufen bzw. mit welchen Parametern sie versorgt werden müssen, können sie den folgenden Beschreibungen entnehmen.

```
Routine  :  InitGels (GelHead,GelTail,GelsInfo) (A0,A1,A2)
Library  :  graphics.library
Offset   :  -120 = -$78
Parameter:  A0 = Zeiger auf einen Puffer mit 58 Null-Bytes
            A1 = Zeiger auf einen Puffer mit 58 Null-Bytes
            A2 = Zeiger auf angelete `GelsInfo`-Struktur
Rückgabe :  keine
Erklärung:  Diese Routine initialisiert das Gelsystem. Die `GelsIn-
            fo`-Struktur braucht dann nur noch mit der `Rast-
            Port`-Struktur verbunden werden (Offset 20)!
```

```
Routine  :  InitMasks (VSprite) (A0)
Library  :  graphics.library
Offset   :  -126 = -$7e
Parameter:  A0 = Zeiger auf angelegte `VSprite`-Struktur
Rückgabe :  keine
Erklärung:  Diese Routine initialisiert die Variablen `CollMask`,
            `BorderLine`, und `ImageShadow`. Es muß natürlich vorher
            genügend Speicher reserviert worden sein.
```

```
Routine  :  AddBob (BOB,RastPort) (A0,A1)
Library  :  graphics.library
Offset   :  -96 = -$60
Parameter:  A0 = Zeiger auf angelegte `BOB`-Struktur
            A1 = Zeiger auf `RastPort`-Struktur
Rückgabe :  keine
```

Erklärung: Diese Routine führt einen BOB zum Gelsystem hinzu. `BOB`- und `VSprite`-Struktur müssen vorher miteinander verbunden werden!

Routine : RemIBob (BOB,RastPort,ViewPort) (A0,A1,A2)
Library : graphics.library
Offset : -132 = -$84
Parameter: A0 = Zeiger auf angelegte `BOB`-Struktur
A1 = Zeiger auf `RastPort`-Struktur
A2 = Zeiger auf `ViewPort`-Struktur
Rückgabe : keine
Erklärung: Diese Routine entfernt einen BOB für immer aus dem Gelsystem!

Routine : SortGList (RastPort) (A1)
Library : graphics.library
Offset : -150 = -$96
Parameter: A1 = Zeiger auf `RastPort`-Struktur
Rückgabe : keine
Erklärung: Diese Routine sortiert das Gelsystem (Position etc.). Muß vor jedem `DrawGList ()` aufgerufen werden!

Routine : DrawGList (RastPort,ViewPort) (A1,A0)
Library : graphics.library
Offset : -114 = -$72
Parameter: A0 = Zeiger auf `ViewPort`-Struktur
A1 = Zeiger auf `RastPort`-Struktur
Rückgabe : keine
Erklärung: Diese Routine zeichnet alle BOBs auf den Bildschirm bzw. in die Bitmap!

Auch hier darf natürlich das Beispielprogramm nicht fehlen. Es erzeugt einen einfachen BOB, welcher sich über den Bildschirm bewegt.

```
;------------------------ Programmname = MoveBOB -------------------
;--------------------- rechte Maustaste = Prg.-Ende -----------------

 move.l 4,a6
 move.l #gfxname,a1
 clr.l d0
 jsr -552(a6)                         ; Open Gfx.-Library
 move.l d0,gfxbase

 move.l 4,a6
 move.l #intname,a1
 clr.l d0
 jsr -552(a6)                         ; Open Int.-Library
 move.l d0,intbase

 move.l #newscreen.a0
 move.l intbase,a6
 jsr -198(a6)                         ; Open Newscreen
 move.l d0,screenhd

 move.l d0,a0                         ; Screenadresse nach A0
 add.l #84,a0                         ; plus 84
 move.l a0,rastport                   ; Save Rastportadresse
 move.l d0,a0                         ; Screenadresse nach A0
 add.l #$2c,a0                        ; plus 44
 move.l a0,viewport                   ; Save Viewportadresse
```

```
;---------- Gels intitialisieren ----------

 move.l #dummyhead,a0
 move.l #dummytail,a1
 move.l #gelsinfo,a2                ; Zeiger auf GelsInfo-Struktur
 move.l gfxbase,a6                  ; gfxbase
 jsr -120(a6)                       ; InitGels ()

 move.l rastport,a0                 ; Zeiger auf RastPort nach A0
 move.l #gelsinfo,20(a0)            ; Gelsinfo in Rastportstruktur

 move.l gfxbase,a6                  ; gfxbase
 move.l #vsprite,a0                 ; Zeiger auf VSprite-Struktur
 jsr -126(a6)                       ; InitMasks ()

 move.l #bob,a0                     ; Zeiger auf BOB-Struktur
 move.l rastport,a1                 ; Zeiger auf RastPort-Struktur
 move.l gfxbase,a6                  ; gfxbase
 jsr -96(a6)                        ; AddBOB ()

draw_loop:
 move.l rastport,a1                 ; Zeiger auf RastPort
 move.l gfxbase,a6                  ; gfxbase
 jsr -150(a6)                       ; SortGList ()
 move.l rastport,a1                 ; Zeiger auf RastPort
 move.l viewport,a0                 ; Zeiger auf ViewPort
 move.l gfxbase,a6                  ; gdxbase
 jsr -114(a6)                       ; DrawGList ()
wait:
 btst #10,$dff016                   ; rechte Maustaste gedrückt ?
 beq exit                           ; wenn ja, Programm beenden
 move.l gfxbase,a6
 jsr -270(a6)                       ; WaitTof - damit BOB nicht so
                                    ; flackert
 bsr move_bob                       ; BOB über Bildschirm bewegen
 bra draw_loop                      ; das ganze wieder von vorne
```

```
;---------- BOB über den Bildschirm bewegen ----------

move_bob:
 move.l #vsprite,a0
 move.w 24(a0),d0                 ; X-pos.
 cmp.w #300,d0                    ; schon bei 300 ?
 bhi aneg                         ; wenn ja, dann andere Richtung
 cmp.w #0,d0                      ; schon bei 0 ?
 bhi noneg                        ; wenn nicht, dann weiter

aneg:
 neg.w speedx                     ; Richtung wechseln

noneg:
 add.w speedx,d0                  ; BOB in X-Richtung bewegen
 move.w d0,24(a0)                 ; neue x-position
 move.w 22(a0),d0                 ; Y-pos.
 cmp.w #184,d0                    ; schon bei 184 ?
 bhi aneg2                        ; wenn ja, dann andere Richtung
 cmp.w #0,d0                      ; schon bei 0 ?
 bhi noneg2                       ; wenn nicht, dann weiter

aneg2:
 neg.w speedy                     ; Richtung wechseln

noneg2:
 add.w speedy,d0                  ; BOB in Y-Richtung bewegen
 move.w d0,22(a0)                 ; neue y-pos.
 rts

;---------- Programm beenden ----------

exit:
 move.l intbase,a6
 move.l screenhd,a0
 jsr -66(a6)                      ; Close Newscreen
```

```
 move.l intbase,a1
 move.l 4,a6                         ; Close Int.-Library
 jsr -414(a6)

 move.l gfxbase,a1
 move.l 4.a6
 jsr -414(a6)                        ; Close Gfx.-Library

 rts                                 ; Ende

;---------- Parameter ----------

intname: dc.b "intuition.library",0
 even
gfxname: dc.b "graphics.library",0
 even
intbase: dc.l 0
gfxbase: dc.l 0
screenhd: dc.l 0

newscreen:
 dc.w 0,0,320,256.5.$0103,2,15
 dc.l 0,title,0,0
title: dc.b "BOB-Demo",0
 even
rastport: dc.l 0
viewport: dc.l 0

;---------- Bob-Parameter ----------

dummyhead: blk.b 58,0
dummytail: blk.b 58,0
gelsinfo: blk.b 38,0

speedx: dc.w 2
speedy: dc.w 2
```

```
vsprite:                            ; ab hier VSprite-Struktur
 dc.l 0,0                           ; Next/Prev-VSprite
 dc.l 0,0                           ; Draw/Clear-Path
 dc.w 0,0                           ; Old Y,X
 dc.w 6                             ; Flag: SaveBack/Overlay
 dc.w 40,80                         ; Y,X-Position
 dc.w 16,1                          ; Höhe = 16 und Breite = 1 Word
 dc.w 2                             ; Tiefe = 2 BitMaps
 dc.w 0,0                           ; MeMask/HitMask
 dc.l imagedata                     ; Grafikdaten
 dc.l borderline                    ; Borderline
 dc.l imageshadow                   ; CollMask = ImageShadow
 dc.l 0                             ; SprColors
 dc.l bob                           ; Zeiger auf BOB-Struktur
 dc.b 3,0                           ; PlanePick/PlaneOnOff

bob:                                ; ab hier BOB-Struktur
 dc.w 0                             ; Flag: normal
 dc.l savebuffer                    ; Puffer für Hintergrundspeiche-
                                    ; rung
 dc.l imageshadow                   ; Maskenpuffer
 dc.l 0,0                           ; Priorität egal
 dc.l vsprite                       ; Zeiger auf dazugehörige VSpri-
                                    ; te-Struktur
 dc.l 0,0                           ; keine Animation und double-
                                    ; buffering
;imagedata:
 dc.w 0,0,0,0,$c60,$c60,$c60,$600c,$600c,$3018,$3018,$1c70,$fe0,$380
 dc.w 0,0,$7c0,$1ff0,$3ff8,$7ffc,$7ffc,$fffe,$fffe,$fffe,$fffe,$fffe
 dc.w $7ffc,$7ffc,$3ff8,$1ff0,$7c0,$0

savebuffer: blk.b 16*1*2*5,0
imageshadow: blk.b 16*1*2,0
borderline: blk.b 1*2,0          ; Size = Bobbreite*2
 END
;------------------------- Listingende -------------------------
```

8.5 Double-Buffer-BOBs

BOBs ab einer bestimmten Größe, fangen, wenn man sie über den Screen bewegt, an zu flackern. Dies wird dadurch verursacht, wenn man mit dem Blitter eine Operation startet, also BOBs bewegt, das der Rasterstrahl z. B. gerade die 150. Zeile durchläuft aber der Blitter soll Daten bewegen, die schon ab Zeile 100 beginnen und bis Zeile 200 liegen. Dann werden nähmlich erst alle Daten ab Zeile 150 dargestellt und die Daten die ab Zeile 100 liegen, werden erst beim nächsten Rasterstrahldurchlauf dargestellt. Wenn man nicht allzuviele, übermäßig Große Objekte bewegen möchte, gibt es die Möglichkeit mit der Routine 'WaitTOF ()' auf die Vertikale Austastlücke zu warten und danach sollte man dann seine Objekte bewegen. Oder man wartet mit der Routine 'VBeamPos ()' bis die letzte Zeile des Screens aufgebaut wurde und bewegt dann seine Objekte.

Möchte man aber viele und große Objekte flackerfrei bewegen, so geht dieses nur im DoubleBuffering-Modus. Dies klingt komplizierter als es ist.

Man legt dazu einfach zwei gleichgroße Bitmaps an, also zwei Bitmapstrukturen. Wenn man die Objekte bewegen möchte, so wird dieses im unsichtbaren, nicht aktiven Display, mit den Routine 'SortGList ()' und 'DrawGList ()' erledigt, schaltet dann die unsichtbare Bitmapstruktur in die Rasterportstruktur, wartet auf die Vertikale Austastlücke, berechnet dann mit 'MakeVPort ()', 'MrgCop ()' und 'LoadView ()' die Copperlisten neu.

Dazu muß aber jedesmal der Hintergrund von zwei Bitmaps gerettet werden. Aber woher weiß das System, wo nun die Adressen der beiden Displays liegen? Dazu gibt die Struktur 'DBufPacket'. Diese Struktur initialisiert man, in dem man den Zeiger "DBuffer" aus der BOB-Struktur auf diese Struktur zeigen läßt. Wenn man diesen Zeiger auf Null setzt, läuft der BOB automatisch

nicht im DoubleBuffering-Modus. Außerdem kann man entweder nur alle BOBs im DoubleBuffering laufen lassen, oder keinen. Wie die DBufPacket-Struktur aussieht, wird nun im folgenden Beschrieben:

DBufPacket-Struktur: (Länge = 16 Bytes)

Offset	Typ	Bezeichnung	Beschreibung
000	Word	BufY	Y-Position vom Hintergrund des zweiten Screens
002	Word	BufX	X-Position vom Hintergrund des zweiten Screens
004	Long	BufPath	Zeiger auf VSprite-Struktur
008	Long	BufBuffer	Zeiger auf einen Puffer, der genausogroß ist, wie der von SaveBuffer, zum Speichern des zweiten Hintergrundes
012	Long	BufPlanes	Zeiger auf Planepointers des zweiten Screens
016		END	Ende der DBufPacket-Struktur

Bei dieser Struktur braucht bloß der BufBuffer-Zeiger initialisiert werden, die anderen Variablen werden automatisch vom System beschrieben! Der "BufBuffer" ist genauso groß, wie der "SaveBuffer"-Puffer von der BOB-Struktur und muß genauso von Ihnen reserviert werden.

Um ein DoubleBuffering-BOB über den Bildschirm zu bewegen, sind folgende Schritte durchzuführen:

1. zwei gleichgroße Bitmaps erstellen und DBufPacket-Struktur init.

Loop: a. unsichbare Bitmapstruktur in die Raster-

Port-Struktur einhängen

b. BOB bewegen (Animate () oder Koordinaten ändern)

c. 'SortGList ()' und 'DrawGList ()' aufrufen

d. 'WaitTOF ()' aufrufen e. Copperlisten neuberechnen [entweder mit 'MakeVPort ()', 'MrgCop ()' und 'LoadView ()' oder bei Screens mit 'RemakeDisplay ()']

f. Punkt a durchführen JMP Loop

Das "double-buffering" findet beim Amiga große Anwendungsvielfalt. Es wird in jedem Spiel, bei Demos oder in Malprogrammen eingesetzt. Im Kapitel "BOB-Animation" ist ein Beispielprogramm vorhanden, was BOBs in diesem Modus bewegt.

8.6 BOB-Animation

Der Amiga bietet die Möglichkeit Objekte automatisch vom System animieren zu lassen. Man kann sogar verschiedene Animationssequencen als ein Objekt darstellen. Damit wird es spielend einfach bewegete Figuren, wie wie zum Beispiel einen laufenden Hund, einen fliegenden Hubschrauber etc. zu erzeugen.

Ein solcher BOB wird als Animationsobjekt bezeichnet. Das Animationsobjekt besteht aus verschiedenen Animationskomponenten (Sequencen). Ein Animationskomponent (Sequence) wird wiederum durch einen Bob dargestellt.

Folgendes Schaubild läßt sich dadurch erstellen:

'AnimOb'-Struktur	; Verbindet einzelne Sequencen (AnimComps) zu einem Objekt
'AnimComp'-Struktur	; Stellt eine Sequence dar (Komponent)

'Bob'-Struktur ; Bestimmt das Aussehen der einzelnen Komponenten
'VSprite'-Struktur ; wird immer mit einer BOB-Struktur initialisiert

Um ein Objekt darzustellen muß das BOBisComp-Flag in den BOB-Flags gesetzt (Bit 1 - siehe BOB-Struktur) und mindestens die 4 oben aufgeführten Strukturen initialisiert werden. Mit 'AddAnimOb ()' , kann das Objekt ins System eingebunden werden. Nun kann man mit der Routine 'Animate ()' das Objekt vom System animieren lassen und mit 'SortGList ()' und 'DrawGList ()' auf den Bildschirm bringen.

Es folgt nun ein Beschreibung der eben genannten Routinen:

```
Routine   :  AddAnimOb (AnimOb,HeadOb,RastPort) (A0,A1,A2)
Library   :  graphics.library
Offset    :  -156 = -$9c
Parameter:   A0 =  Zeiger auf angelegte `AnimOb`-Struktur
             A1 =  Zeiger auf einen Puffer von 4 Bytes (dc.l 0). Die-
                   ser Puffer zeigt immer auf das zuletzt eingefügte
                   `AnimOb`. Beim ersten Aufruf dieser Routine, muß
                   dieser Puffer mit Null gefüllt sein.
             A2 =  Zeiger auf `RastPort`-Struktur
Rückgabe :   keine
Erklärung:   Diese Routine führt ein Objekt zum Gelsystem hinzu.
```

```
Routine   :  Animate (HeadOb,RastPort) (A0,A1)
Library   :  graphics.library
Offset    :  -162 = -$a2
Parameter:   A0 =  Zeiger auf einen Puffer von 4 Bytes (dc.l 0). Die-
                   ser Puffer zeigt immer auf das zuletzt eingefügte
                   `AnimOb`. (s.o.)
             A1 =  Zeiger auf `RastPort`-Struktur
```

```
Rückgabe :  keine
Erklärung:  Diese Routine animiert ein Objekt. Es berechnet entspre-
            chent die Werte in den Strukturen neu. Mit `SortGList ()`
            und `DrawGList ()` kann man dann das Objekt auf den
            Bildschirm darstellen.
```

Um ein 'AnimObjekt' über den Bildschirm bewegen zu können, müssen folgende Schritte eingehalten werden:

1.	jsr 'Animate ()'	; Anim-Strukturen berechnen
2.	jsr 'SortGList ()'	; Gel-Liste sortieren
2.1	jsr 'DoCollision ()'	; wenn Collision noch abgefragt werden soll
3.	jsr 'DrawGlist ()'	; Gel-Liste auf Bildschirm ausgeben

Es fehlt jetzt nur noch die Beschreibung der beiden Strukturen 'AnimOb'- und 'AnimComp'- , welche jetzt folgt:

'AnimOb'-Struktur: (Länge = 40 Bytes)

Die AnimOb-Struktur enthält ein Animationsobjekt, welches aus verschiedenen AnimComps besteht, in seiner Gesamtheit. Dieses Objekt wird mit 'AddAnimOb ()' dem System zugänglich gemacht und kann mit 'Animate ()' animiert werden. 'SortGList ()' und 'DrawGList ()' printen dann das Objekt auf den Bildschirm.

Offset:	**Typ:**	**Bezeichnung:**	**Erklärung:**
;---- es folgen als erstes Systemvariablen -----			
000	Long	NextOb	Zeiger auf nächste 'AnimOb'-Struktur (oder Null)

004	Long	PrevOb	Zeiger auf Vorgängerstruktur (s.o.)
008	Long	Clock	enthält Anzahl 'Animate ()'-Aufrufe die dieses Objekt schon erlebt hat
012	Word	AnOldY	alte y-Position
014	Word	AnOldX	alte x-Position

;--- nun folgen die User-Variablen -----

016	Word	AnY	y-Position (Achtung - siehe unten)
018	Word	AnX	x-Position (Achtung - siehe unten)
020	Word	YVel	Geschwindigkeit in y-Richtung (siehe unten)
022	Word	XVel	Geschwindigkeit in x-Richtung (siehe unten)
024	Word	YAccel	y-Beschleunigung (siehe unten)
026	Word	XAccel	x-Beschleunigung (siehe unten)
028	Word	RingYTrans	steigende Beschleunigung in Y-Richtung (s. u.)
030	Word	RingXTrans	steigende Beschleunigung in X-Richtung (s. u.)
032	Long	AnimORoutine	siehe unten
036	Long	HeadComp	Zeiger auf erste 'AnimComp'-Struktur
040		END	Ende der 'AnimOb'-Struktur

Erklärungen:

AnY,AnX (Offset 016 und 018)

Aktuelle Position des AnimObs. Wird nicht in Punkten bzw. Zeilen angegeben, sondern aufgrund der Geschwindigkeits- und Beschleunigungsvariablen, in 64stel Schritten. Das heißt, man muß jeden Wert mit 64 multiplizieren und das Ergebnis kann dann in diesen Variablen eingetragen werden.

YVel,XVel (Offset 020 und 022)

Diese Werte werden bei jedem Aufruf von 'Animate ()' zu AnX und AnY addiert. Sie werden normal, nicht in 64stel Schritten angegeben. Sie geben die Anzahl Punkte an, um die ein Objekt bewegt werden soll. Es sind die Geschwindigkeiten der Objekte.

YAccel,XAccel (Offset 024 und 026)

Diese Werte werden zu YVel und XVel addiert. Sie werden auch normal, und nicht in 64stel Schritten angegeben. Es ist die Beschleunigung der Objekte.

RingYTrans,RingXTrans (Offset 028 und 030)

Werte werden direkt zu AnX und AnY addiert. Wenn man diese Beschleunigungsart wählt, muß man YVel, XVel, YAccel und XAccel mit Null initialisieren.

AnimORoutine (Offset 032)

Die Routine, deren Adresse hier angegeben wird, wird bei jedem 'Animate ()'- Aufruf einmal angesprungen. Man kann so die Positionen oder dergleichen testen. Die Routine muß mit einem "RTS" enden.

HeadComp (Offset 036)

Hier muß die Adresse der ersten AnimComp-Struktur (Sequence) eingetragen werden. Woher sollte sonst das System wissen, aus welchen und aus wievielen Sequencen das Objekt besteht.

Die AnimComp-Struktur (Animationskomponente) stellt die Verbindung zwischen BOB und Animationsobjekt (AnimOb-Struktur) her. Besonders bei der Sequenz- animation (RingTrigger) ist sie von gro°er Wichtigkeit, denn sie verbindet einzelne Komponenten zu einem Ring miteinander und bestimmt, wie lange jede einzelne Sequenz aktiv sein soll.

AnimComp-Struktur: (Länge = 38 Bytes)

Offset:	Typ:	Bezeichnung:	Erklärung:
000	Word	CompFlags	Bit 0 = 1 , dann wird mit Hilfe der Zeiger 'NextSeq' und 'PrevSeq' eine Ringsequenz vom System durchgeführt. (RingTrigger-Flag)
002	Word	Timer	wird mit Wert aus TimeSet geladen und auf Null heruntergezählt und dann falls gewünscht

			(RingTrigger-Flag = 1) die nächste Sequenz dargestellt
004	Word	TimeSet	wie lange eine Sequenz dargestellt werden soll
006	Long	NextComp	Zeiger auf nächste AnimComp-Struktur (s. u.)
010	Long	PrevComp	Zeiger auf Vorgänger-AnimComp-Struktur (s. u.)
014	Long	NextSeq	Zeiger auf nächste AnimComp-Struktur (s. u.)
018	Long	PrevSeq	Zeiger auf Vorgänger-AnimComp-Struktur (s. u.)
022	Long	AnimCRoutine	siehe 'AnimORoutine'
026	Word	Y-Trans	y-Positon der AnimComp-Struktur, in 64stel
028	Word	X-Trans	X-Postion der AnimComp-Struktur, in 64stel
030	Long	HeadOb	Zeiger auf dazugehörige 'AnimOb'-Struktur
034	Long	AnimBob	Zeiger auf dazugehörige BOB-Struktur
038		END	Ende der AnimComp-Struktur

Erklärungen:

CompFlags (Offset 000)

Wird hier Bit 0 = 1 gesetzt, wird eine Ringsequence erzeugt. Das heißt, um zum Beipiel die Beinbewegung einer Figur in bestimmten Zeitabständen ablaufen lassen zu können, wird die Timer-Variable mit der gewünschten Zeit gespeist und NextSeq/PrevSeq auf die einzelnen Sequencen eingestellt. Nachdem die letzte Sequence dann dargestellt wurde, beginnt das System automatisch wieder mit der Ersten.

Timer (Offset 002)

Wird für die Ringsequence benötigt. Diese Variable wird mit "TimeSet" geladen. Und nach jedem "Animate ()" Aufruf, einen heruntergezählt. Ist der Timer bei Null angelangt, wird die nächste Sequence dargestellt.

TimeSet (Offset 004)

Hier steht der Wert, mit dem der Timer geladen wird. Also wieviel "Animate ()"-Aufrufe nötig sind, bis die nächste Sequence dargestellt wird.

NextComp,PrevComp (Offset 006 und 010)

Mit diesen beiden Variablen ist es möglich, innerhalb eines Animationsobjektes (AnimOB) mehrere Komponenten (z. B. Arme, Beine, Kopf eines Männchens) miteinander zu verbinden, die dann durch 'Animate ()' dargestellt werden. Hier müssen dann die Adressen der entsprechenden AnimComp-Strukturen eingetragen werden.

NextSeq,PrevSeq (Offset 014 und 018)

Mit diesen beiden Zeigern, lassen sich z. B. die einzelnen Bewegungen (Sequenzen) eines Armes darstellen, die dann durch 'Animate ()' dargestellt werden. Das 'RingTrigger'-Flag muß dazu in den CompFlags gesetzt sein. 'PrevSeq' des ersten 'AnimComp' muß auf den letzen 'AnimComp' zeigen. 'NextSeq' des letzten 'AnimComp' muß auf den ersten 'AnimComp' zeigen. Diese Schritte sind einzuhalten, damit es zu einer Ringsequence kommen kann.

Y-Trans,X-Trans (Offset 026 und 028)

Enthalten die Koordinaten der Sequence. Sie werden relativ zu "AnX" und "AnY" (AnimOb-Struktur) berechnet. Sie werden auch in 64stel-Schritten angegeben. Es kann zum Beispiel auftreten, das die BitMap so groß ist, daß die Words in dem die Koordinaten stehen nicht mehr ausreichen, um diese Werte zu fassen. Um größere Positonen bei 'AnX' und 'AnY' zu erreichen, setzt man den Mehrwert dann einfach in 'XTrans' und 'YTrans'.

HeadOb (Offset 030)

Hier muß die Adresse der dazugehörigen AnimOb-Struktur stehen, weil sonst das System nicht weiß, zu welchem Objekt nun diese Sequence gehört.

AnimBob (Offset 034)

Hier müssen sie die Adresse der BOB-Struktur eintragen, welche die entsprechende Grafik für die Sequence enthält.

Das nachfolgende Beispiel-Listing erzeugt einen BOB im "double-bouffering" Modus, der vom System animiert wird. Mit der rechten Maustaste kann das Programm wieder beendet werden. Es wurde auch kein Screen, sondern ein primitives Grafik-Display über die Grafik-Routinen erzeugt.

```
;--------------------- Programmname = AnimateBOB -------------------
;---------------------- rechte Maustaste = Ende ---------------------
;------------------ erzeugt einen double-bouffering BOB -------------
;--------------------- der vom System animiert wird -----------------

 breite = 320              ; Bitmapbreite (muß durch 8 teilbar sein)
 höhe = 256                ; Bitmaphöhe
 tiefe = 5                 ; Anzahl Bitplanes

;---------- Librarys öffnen ----------

 bsr openlibrarys

;---------- Speicher für Bitmaps reservieren ----------

 move.l #((breite/8)*tiefe*höhe*2),d0     ; 2 gleichgroße Bitmaps
 move.l #$10002,d1                        ; Chip+ClearMEM
 move.l 4,a6                              ; execbasis
 jsr -198(a6)                             ; AllocMem ()
 move.l d0,displaybase                    ; Basis speichern
 tst.l d0                                 ; wenn Error,
 beq exit                                 ; dann Exit

;---------- Bitmap Pointer errechnen 2x ----------

 move.l displaybase,a0                 ; Basis des Display
 move.l #bitmap0,a1                    ; Bitmap0-Adresse
 bsr search_ptr                        ; BitmapPtr-Adressen errechnen
 move.l displaybase,a0                 ; Basis des Display
 add.l #((breite/8)*höhe*tiefe),a0     ; plus einer Screengröße
 move.l #bitmap1,a1                    ; Bitmap1-Adresse
 bsr search_ptr                        ; BitmapPtr-Adressen errechnen
```

```
;---------- alten View speichern ----------

 move.l intbase,a6                    ; intutionbasis
 jsr -294(a6)                         ; ViewAdress ()
 move.l d0,oldview                    ; alten View speichern

;---------- View-Struktur init. ----------

 move.l gfxbase,a6                    ; gfxbasis
 move.l #view,a1                      ; Zeiger auf neue View-Struktur
 jsr -360(a6)                         ; InitView ()

;---------- ViewPort-Struktur init. ----------

 move.l gfxbase,a6                    ; gfxbasis
 move.l #viewport,a0                  ; Zeiger auf neue Viewport-
                                      ; Struktur
 jsr -204(a6)                         ; InitVPort ()

;---------- Viewport-Parameter init. ----------

 move.w #breite,viewport+24           ; Breite vom ViewPort
 move.w #höhe,viewport+26             ; Höhe vom ViewPort

;---------- Bitmap-Struktur init. 0/1 ----------

 move.l gfxbase,a6                    ; gfxbase
 move.l #bitmap0.a0                   ; Bitmap0-Struktur
 move.l #tiefe,d0                     ; Depth
 move.l #breite,d1                    ; width
 move.l #höhe,d2                      ; heigth
 jsr -390(a6)                         ; InitBitMap ()
 move.l gfxbase,a6                    ; gfxbasis
 move.l #bitmap1,a0                   ; Bitmap1-Struktur
 move.l #tiefe,d0                     ; Depth
 move.l #breite,d1                    ; Width
```

```
 move.l #höhe,d2                     ; Heigth
 jsr -390(a6)                        ; InitBitMap ()

;---------- Rasterport init. ----------

 move.l #rastport,a1                 ; Zeiger auf neue RastPort-
                                     ; Struktur
 move.l gfxbase,a6                   ; gfxbasis
 jsr -198(a6)                        ; InitRastPort ()

;---------- ColorMap-Struktur vom System anlegen lassen ----------

 move.l gfxbase,a6                   ; gfxbasis
 move.l #32,d0                       ; 32 Farben
 jsr -570(a6)                        ; GetColorMap ()
 move.l d0,colormap                  ; Adresse speichern
 tst.l d0                            ; wenn Error,
 beq exit                            ; dann Exit

;---------- Strukturen miteinander verbinden ----------

 move.l #rasinfo,viewport+36         ; RasInfo in ViewPort
 move.l colormap,viewport+4          ; Colormap in ViewPort
 move.l #bitmap0,rasinfo+4           ; Bitmap0 in RasInfo
 move.l #bitmap0,rastport+4          ; Bitmap0 in RastPort
 move.l #viewport,view               ; ViewPort in View

;--------------------------------------------------------------------

;---------- Neue Copperliste erzeugen und starten ----------

 jsr copperstart(pc)
 move.w #1,copper                    ; Copperstatus = 1 , es wurden
                                     ; neue Copperlisten erstellt

 bsr initbob                         ; GelSystem initialisieren
```

```
 bsr printtext                       ; Text ausgeben in Bitmap0
 bsr changebitmaps                   ; BitMaps vertauschen
 bsr printtext                       ; Text ausgeben auch in Bitmap1
 bsr changebitmaps
 bsr sortglist                       ; Gel-Liste sortieren
 bsr drawglist                       ; und ausgeben (Start-Positionen)

main:
 btst #10,$dff016                    ; rechte Maustaste ?
 beq exit
 bsr changebitmaps                   ; Bitmaps vertauschen
 bsr move_bob                        ; BOB bewegen
 bsr sortglist                       ; Gel-Liste sortieren
 bsr drawglist                       ; und ausgeben
 bsr no_flackern                     ; auf Austastlücke warten
 bsr copperstart                     ; neue Copperliste aktivieren
 bra main

exit:
 move.l oldview,a1                   ; auf alte Copperlist umschalten
 cmp.l #0,a1                         ; wenn eine vorhanden ist?
 beq exit1                           ; wenn nicht, dann weiter
 move.l gfxbase,a6                   ; alte Copperlist starten
 jsr -222(a6)                        ; LoadView ()

exit1:
 move.l colormap,a0                  ; wurde eine
 cmp.l #0,a0                         ; ColorMap angelegt ?
 beq exit1a                          ; wenn nicht, dann weiter
 move.l gfxbase,a6                   ; ColorMap wieder freigeben
 jsr -576(a6)                        ; FreeColorMap ()
```

```
;---------- Speicher der einzelnen Copperlisten wieder freigeben ----

exit1a:
 cmp.w #0,copper                   ; wurden neue Copperlisten er
                                   ; stellt, und somit
 beq exit1b                        ; Speicher vom System belegt?
 move.l #viewport,a0               ; erzeugte Copperlisten wieder
                                   ; freigeben
 move.l gfxbase,a6
 jsr -540(a6)                      ; FreeVPortCopLists ()
 move.l view+4,a0
 move.l gfxbase,a6
 jsr -564(a6)                      ; FreeCprList ()

exit1b:
 move.l displaybase,a1             ; wurde Speicher
 cmp.l #0,a1                       ; für Bitmaps reserviert ?
 beq exit2                         ; wenn nicht, dann weiter
 move.l #((breite/8)*tiefe*höhe*2),d0          ; Size
 move.l 4,a6                       ; execbasis
 jsr -210(a6)                      ; FreeMem ()

exit2:
 bsr closelibrarys                 ; Librarys schließen
 rts                               ; Ende

;---------- Librarys öffnen ----------

openlibrarys:
 move.l 4,a6
 move.l #gfxname,a1                ; Graphics
 clr.l d0
 jsr -552(a6)                      ; OpenLibrary ()
 move.l d0,gfxbase
 move.l 4,a6
 move.l #intname,a1                ; Intuition
```

```
 clr.l d0
 jsr -552(a6)                    ; OpenLibrary ()
 move.l d0,intbase
 rts

;---------- Librarys schließen ----------

closelibrarys:
 move.l 4,a6
 move.l gfxbase,a1
 jsr -414(a6)                    ; CloseLibary ()
 move.l 4,a6
 move.l intbase,a1
 jsr -414(a6)                    ; CloseLibary ()
 rts

;---------- Einen Text printen, damit man was sieht ----------

printtext:
 move.l #rastport,a0             ; RastPort-Struktur
 move.l #text1,a1                ; Text-Struktur
 move.l #00,d0
 move.l #50,d1
 move.l intbase,a6
 jsr -216(a6)                    ; PrintIText ()
 rts

;---------- 100%iges No Flackern der Bobs ----------

no_flackern:
 move.l gfxbase,a6
 jsr -270(a6)                    ; WaitTOF
 rts
```

```
;---------- Copperlisten konstruieren ----------

copperstart:
 move.l gfxbase,a6                      ; gfxbasis
 move.l #view,a0                        ; View-Struktur
 move.l #viewport,a1                    ; ViewPort-Struktur
 jsr -216(a6)                           ; MakeVport ()

;---------- Copperlisten zu einer Copperliste zusammenfassen -------

 move.l gfxbase,a6                      ; gfxbasis
 move.l #view,a1                        ; View-Struktur
 jsr -210(a6)                           ; MrgCop ()

;---------- Neue Copperliste starten ----------

 move.l gfxbase,a6                      ; gfxbasis
 move.l #view,a1                        ; View-Struktur
 jsr -222(a6)                           ; LoadView ()
 rts

;---------- Gelsystem init. ----------

initbob:
 move.l #dummy1,a0                      ; Dummy1
 move.l #dummy2,a1                      ; Dummy2
 move.l #gelsinfo,a2                    ; GelsInfo-Struktur
 move.l gfxbase,a6                      ; gfxbasis
 jsr -120(a6)                           ; InitGels ()

 move.l #rastport,a0
 move.l #gelsinfo,20(a0)                ; Gelsystem in Rastport

 move.l #vsprite,a0                     ; VSprite-Struktur
 move.l gfxbase,a6                      ; gfxbasis
 jsr -126(a6)                           ; InitMasks ()
```

```
 move.l #animob,a0                        ; AnimOB-Struktur
 move.l #headob,a1                        ; Zeiger auf letztes AnimOB
 move.l #rastport,a2                      ; RastPort-Struktur
 move.l gfxbase,a6                        ; gfxbasis
 jsr -156(a6)                             ; AddAnimOb ()
 rts

;---------- Bob bewegen ----------

move_bob:
 move.l #animob,a0                        ; AnimOb-Strukur
 move.w 18(a0),d0                         ; AnX-Position
 cmp.w #320*64,d0                         ; schon bei 320 ?
 bpl aneg                                 ; wenn ja, dann Richtungswechsel
 cmp.w #-10*64,d0                         ; schon bei -10 ?
 bpl noneg                                ; wenn nicht, dann weiter

aneg:
 neg.w 22(a0)                             ; XVel = Vorzeichen wechseln

noneg:
 move.w 16(a0),d0                         ; AnY-Position
 cmp.w #256*64,d0                         ; schon bei 256 ?
 bpl aneg2                                ; wenn ja, Richtungswechsel
 cmp.w #-10*64,d0                         ; schon bei -10 ?
 bpl noneg2                               ; wenn nicht, weiter

aneg2:
 neg.w 20(a0)                             ; YVel = Vorzeichen wechseln

noneg2:
 move.l #headob,a0                        ; Zeiger auf letztes AnimOb
 move.l #rastport,a1                      ; RastPort-Struktur
 move.l gfxbase,a6                        ; gfxbasis
 jsr -162(a6)                             ; Animate ()
 rts
```

```
;---------- Displays im Rasterport vertauschen ----------

changebitmaps:
 move.l rastport+4,a0                 ; Bitmap-Struktur aus RastPort
 cmp.l #bitmap0,a0                    ; liegt Bitmap0 vor ?
 bne no                               ; wenn nicht, dann no
 move.l #bitmap1,rastport+4           ; sonst Bitmap1-Struktur
 move.l #bitmap1,rasinfo+4            ; aktivieren
 rts

no:
 move.l #bitmap0,rastport+4           ; sonst Bitmap0-Struktur
 move.l #bitmap0,rasinfo+4            ; aktivieren
 rts

;---------- Bobs anzeigen ----------

sortglist:
 move.l #rastport,a1                  ; RastPort-Struktur
 move.l gfxbase,a6                    ; gfxbasis
 jsr -150(a6)                         ; SortGList ()
 rts

drawglist:
 move.l #rastport,a1                  ; RastPort-Struktur
 move.l #viewport,a0                  ; ViewPort-Struktur
 move.l gfxbase,a6                    ; gfxbasis
 jsr -114(a6)                         ; DrawGList ()
 rts
```

```
;---------- Bitmappointer errechnen ----------

                                        ; A0 = Zeiger auf freien Speicher
                                        ; A1 = Zeiger auf Bitmapxstruk.
search_ptr:
 move.l #breite/8.d0
 mulu #höhe,d0                          ; In D0 = größe einer Bitmap
 move.w #tiefe-1,d1                     ; In D1 = Anzahl Bitmaps
 move.l #8,d2

Ptr_loop:
 move.l a0,(a1,d2)                      ; Ptr. auf Bitmaps in Bit-
                                        ; map0struk. saven
 add.l d0,a0                            ; eine Map weiter gehen
 add.l #4,d2                            ; Pointer-Offset plus 4
 dbra d1,Ptr_loop                       ; Schleife bis alle Pointer durch
 rts

;---------- Parameter ----------

gfxname: dc.b "graphics.library",0
 even
intname: dc.b "intuition.library",0
 even
gfxbase: dc.l 0                         ; Graphics-Basis
intbase: dc.l 0                         ; Intuition-Basis
oldview: dc.l 0                         ; alte View-Adresse
colormap: dc.l 0                        ; Zeiger auf ColorMap-Struktur

view: blk.b 18,0                        ; View-Struktur
viewport: blk.b 40,0                    ; ViewPort-Struktur
rasinfo: blk.b 12,0                     ; RasInfo-Sturktur
rastport: blk.b 100,0                   ; RastPort-Struktur
bitmap0: blk.b 40.0                     ; Bitmap0-Struktur
bitmap1: blk.b 40.0                     ; Bitmap1-Struktur
```

```
copper: dc.w 0
displaybase: dc.l 0
text1:
 dc.b 1.0,0,0
 dc.w 0.0
 dc.l 0,title,0
title: dc.b "DoubleBuffer BOB (rechte Maustaste)",0
 even
speedx: dc.w 1
speedy: dc.w 1
buffer: dc.l 0
headob: dc.l 0            ; Adresse des zuletzt eingefügten AnimOb
dummy1: blk.b 58,0
dummy2: blk.b 58,0
gelsinfo:                           ; ab hier GelsInfo-Struktur
 dc.b 0,0
 dc.l 0,0
 dc.l nextline
 dc.l lastcolor
 dc.l collhandler
 dc.w 0,0,0,0
 dc.l 0,0
nextline: blk.w 8.0
lastcolor: blk.w 32.0
collhandler: blk.l 16.0

animob:                             ; ab hier AnimOb-Struktur
 dc.l 0.0                           ; NextOb/PrevOb
 dc.l 0                             ; Clock
 dc.w 0.0                           ; AnOldY/AnOldX
 dc.w 64*10,64*20         ; AnY.AnX - Koordinaten vom Objekt
 dc.w 35,35               ; YVel,XVel - Geschwindigkeit
 dc.w 0,0                 ; YAccel,XAccel - keine Beschleunigung
 dc.w 0.0                 ; RingYTrans,RingXTrans
 dc.l 0                   ; keine AnimORoutine
 dc.l animcomp            ; HeadComp - Zeiger auf AnimComp-Struktur
```

```
animcomp:                ; ab hier AnimComp-Struktur
 dc.w 0                  ; CompFlags = keine RingSequence
 dc.w 0,0                ; Timer,TimeSet = Null
 dc.l 0,0                ; NextComp,PrevComp
 dc.l 0,0                ; NextSeq,PrevSeq
 dc.l 0                  ; AnimCRoutine = keine
 dc.w 0,0                ; Y-Trans,X-Trans
 dc.l animob             ; Zeiger auf dazugehörige AnimOb-Struktur
 dc.l bob                ; Zeiger auf dazugehörige BOB-Struktur

bob:                     ; ab hier BOB-Struktur
 dc.w 2                  ; Flag: BOBisComp
 dc.l savebuffer         ; Puffer für Hintergrundspeicherung
 dc.l imageshadow        ; Maskenpuffer
 dc.l 0,0                ; Priorität egal
 dc.l vsprite            ; Zeiger auf dazugehörige VSprite-Struktur
 dc.l animcomp           ; Zeiger auf dazugehörige AnimComp-Struktur
 dc.l dbuffer            ; Zeiger auf DBufPacket für double-boufering

vsprite:                 ; ab hier VSprite-Struktur
 dc.l 0,0                ; Next/Prev-VSprite
 dc.l 0,0                ; Draw/Clear-Path
 dc.w 0,0                ; Old Y,X
 dc.w 6                  ; Flag: SaveBack/Overlay
 dc.w 0,0                ; Y,X-Position wird in AnimObStruktur gesetzt
 dc.w 16,1               ; Höhe = 16 und Breite = 1 Word
 dc.w 2                  ; Tiefe = 2 BitMaps
 dc.w 0,0                ; MeMask/HitMask
 dc.l imagedata          ; Grafikdaten
 dc.l borderline         ; Borderline
 dc.l imageshadow        ; CollMask = ImageShadow
 dc.l 0                  ; SprColors
 dc.l bob                ; Zeiger auf BOB-Struktur
 dc.b 3,0                ; PlanePick/PlaneOnOff
```

```
dbuffer:                  ; ab hier DBufPacket-Struktur
 dc.w 0,0
 dc.l 0
 dc.l bufbuffer           ; Puffer zum speichern des zweiten Hinter-
                          ; grundes
 dc.l 0

imagedata:
 dc.w 0.0,0,0,$c60,$c60,$c60.$600c,$600c.$3018,$3018,$1c70,$fe0,$380
 dc.w 0,0,$7c0,$1ff0,$3ff8,$7ffc,$7ffc,$fffe.$fffe,$fffe,$fffe,$fffe
 dc.w $7ffc,$7ffc,$3ff8,$1ff0,$7c0,$0

savebuffer: blk.b 16*1*2*5,0
bufbuffer: blk.b 16*1*2*5,0          ; genauso groß wie SaveBuffer
imageshadow: blk.b 16*1*2,0
borderline: blk.b 1*2,0              ; Size = Bobbreite*2
 END
;--------------------------- Listingende --------------------------
```

8.7 Prioritäten von Grafikobjekten

In unseren Beispiellistings haben bisher immer nur einen BOB erzeugt. Dabei brauchten wir uns um die Priorität nicht zu kümmern. Sobald man aber zwei oder mehr BOBs aufeinmal darstellen will, ist die Priorität schon von Bedeutung.

Denn wenn zwei BOBs bewegt werden und sie sich überlappen, kann es passieren, daß mal der eine oder mal der andere im Vordergrund steht. Und daß sicht nun wirklich nicht besonders schön aus. Stellen sie sich mal vor, zwei Flugzeuge überlappen sich beim vorbeifliegen und dabei tritt dann dieser unschöne Effekt auf. Dieses würde überhaupt nicht der Realität entsprechen.

Die Priorität eines BOBs läßt sich mit den Zeigern "Before" und "After" in der BOB-Struktur einstellen. Diese zeigen dann auf die BOB-Strukturen, die vor und nach diesen BOB bearbeitet werden sollen. Dabei ist zu beachten, daß "After" des ersten BOBs auf den zweiten BOB zeigt und das "Before" vom zweiten BOB auf den ersten BOB zeigt. Und diese Zeiger sind nach "AddBOB ()" zu setzen, da die Routine diese Zeiger löscht.

Die Prioritäten von VSprites stehen aufgrund der Hardware schon fest. Dabei gilt, das Sprite mit der höchsten Priorität befindet sich vor allen anderen. Es kann also nicht von anderen VSprites verdeckt werden. Die Nummer bestimmt dabei die Priorität, die ein VSprite besitzt. Je kleiner die Nummer, desto größer die Priorität. Das Sprite mit der Nummer Null besitzt dementsprechend die höchste Priorität. Danach folgt Sprite 1 usw.

8.8 Collisionsabfrage

Um Kollisionen zwischen zwei Objekten festzustellen, muß man als erstes die beiden Variablen 'Borderline' und 'CollMask' intitialisieren .

In 'Borderline' steht das logisch "ODER" der ersten Zeile eines Grafikobjekts. Sie wird zur schnelleren Collisionskontrolle benötigt. Denn erst wenn durch Borderline eine mögliche Collision festgestellt wurde, wird mit "CollMask" auf eine Collision getestet.

Der Puffer der von ihnen reserviert werden muß, beträgt bei VSprites ein Word, bei BOBs entspricht er der Breite die in "Width" (VSprite- Struktur) steht.

Überall wo ein Punkt in der "CollMask" gesetzt ist, wird eine Collision festgestellt. In der Collisionsmaske wird das logisch "ODER" aller Bitplanes des BOBs abgelegt. Auch dieser Puffer muß von ihnen reserviert werden. Er berechnet sich bei VSprites aus dem Produkt zwischen 'Höhe (Heigth) * 2'. Bei BOBs aus dem Produkt zwischen 'Höhe (Heigth) * Breite (Width) * 2'. Man kann auch den 'CollMask'-Zeiger auf den 'ImageShadow'-Puffer zeigen lassen, denn Beide sind identisch. Außerdem spart man dadurch Speicher. Wollen sie jedoch nachträglich die Collisionsmaske ändern, können sie diesen Trick jedoch nicht anwenden.

Zur Verdeutlichung der Pufferberechnungen, das Ganze nochmal in Kurzform:

a) VSprites: (Ergebnisse in Bytes)

BorderlineSize = 1 Word (2 Bytes) CollMaskSize = Spritehöhe * 2

b) BOBs: (Ergebnisse in Bytes)

BorderlineSize = Bobbreite * 2
CollMaskSize = Bobhöhe * Bobbreite * 2

Durch aufrufen der Routine 'InitMasks ()' wird dann, der von ihnen zuvor reservierte Puffer, mit den entsprechenden Masken gefüllt.

Das System kann zwischen 16 verschiedene Collisionsroutinen unterscheiden. Die Collsionsroutine 0 ist für Randberührungen. Die anderen 15 lassen sich für Collisionen zwischen zwei Objekten nutzen.

Welche Routine nun vom System bei einer Collision angesprungen wird, hängt von den Werten in 'MeMask' und 'HitMask' ab. (Siehe VSprite-Struktur) Jedes Bit in einer der Masken, steht für eine Collisionsroutine.

z. B. Bit0 = Collisionsroutine 0 (Rand-Collision).

Mit 'MeMask' gibt man jetzt jedem Objekt eine Nummer von 0-15, und mit 'HitMask' gibt man die Nummer an, mit welchen Objekt diese Objekt nicht zusammenstoßen darf, denn dann wird die Collsionsroutine mit der Nummer in 'HitMask' angesprungen.

Wenn z. B. der Feind in 'MeMask' Bit 2 = 1 ist, und der Spieler in 'HitMask' Bit 2 gesetzt hat, und diese beiden Objekte stoßen zusammen, kommt es zu einer Collision und die Collisionsroutine 2 wird angesprungen. Für Randcollsionen müssen die Ränder in der 'GelsInfo'-Struktur eingestellt werden (leftmost, rightmost, topmost und bottommost). Außerdem muß der 'CollHandler' auf einen Puffer von 16 Langwrötern zeigen.

Bevor aber eine Collsionsroutine angesprungen werden kann, muß erstmal diese Routine ins System eingebunden werden. Dieses er-

ledigt die Routine 'SetCollision ()' . Und mit der Routine 'DoCollision ()' lassen sich dann Collsionen abfragen. Hier eine Beschreibung der beiden Routienen:

```
Routine   :  SetCollision (Type,Routine,GelsInfo) (D0,A0,A1)
Library   :  graphics.library
Offset    :  -144 = -$90
Parameter:   D0 = Nummer der Collisionsroutine (0-15)
             A0 = Zeiger auf Collisionsroutine die mit einem RTS endet
             A1 = Zeiger auf GelsInfo-Struktur
Rückgabe :   keine
Erklärung:   Diese Routine init. eine Collsionsroutine mit angegebener
             Nummer. Die Adresse der Colisionsroutine wird in den
             `CollHandler`- Puffer eingetragen.
```

```
Routine   :  DoCollision (RastPort) (A1)
Library   :  graphics.library
Offset    :  -108 = -$6c
Parameter:   A1 = Zeiger auf RasterPort-Struktur
Rückgabe :   keine
Erklärung:   Diese Routine testet ob eine Collision stattgefunden hat,
             und springt dann in die entsprechende Collisionsroutine.
             Bei BOB-BOB-Collisionen wird der angesprungenen Routine
             in A1 die Adresse des Oberen VSprites und in A2 die des
             Unteren VSprites übergeben. Bei Randberührungen, wird der
             angesprungenen Collisionsroutine (Nummer Null) in A3 die
             Adresse des VSprites übergeben.
```

Nachdem man mit 'SetCollision ()' die Collsionsroutine eingebunden hat. Und die entsprechenden Varaiblen (MeMask, HitMask, BorderLine, CollMask) init. hat, testet man eine Collision wie folgt:

1. jsr 'SortGList ()'
2. jsr 'DoCollision ()'
3. jsr 'DrawGList ()'

Ich muß sie hier leider darauf hinweisen, daß die 'DoCollision ()'-Routine nicht ganz einwandfrei funktioniert. Jedoch können sie durch mehrmaliges aufrufen der Routine diesen Fehler weitgehenst ausschalten.

Das letzte folgende Listing in diesem Kapitel, zeigt die Programmierung zweier BOBs, die im double-buffering-modus laufen. Wenn beide Objekte miteinander zusammenstoßen, blinkt der Hintergrund auf. Außerdem bekommt jeder BOB eine Priorität zugeteilt. Mit der rechten Maustaste kann das Programm wieder beendet werden. Diesmal werden die BOBs allerdings über einen Screen bewegt. Wodurch das ruckeln noch weiter ausgeschaltet wird.

```
;-------------------- Programmname = BOB-Collision -----------------
;------------------ erzeugt zwei BOBs mit Priorität ----------------
;---------------------- und Collisionsabfrage im -------------------
;-------------------- double-buffering und werden ------------------
;----------------------- vom System animiert -----------------------
;---------------------- rechte Maustaste = Ende --------------------

 breite = 320            ; Bitmapbreite (muß durch 8 teilbar sein)
 höhe = 256              ; Bitmaphöhe
 tiefe = 5               ; Anzahl Bitplanes
```

```
;---------- Librarys öffnen ----------

 bsr openlibrarys

;---------- Speicher für Bitmaps reservieren ----------

 move.l #((breite/8)*tiefe*höhe*2),d0     ; 2 gleichgroße Bitmaps
 move.l #$10002,d1                         ; Chip+ClearMEM
 move.l 4,a6                               ; execbasis
 jsr -198(a6)                              ; AllocMem ()
 move.l d0,displaybase                     ; Basis speichern
 tst.l d0                                  ; wenn Error,
 beq exit                                  ; dann Exit

;---------- Bitmap Pointer errechnen 2x ----------

 move.l displaybase,a0                ; Basis des Display
 move.l #bitmap0,a1                   ; Bitmap0-Adresse
 bsr search_ptr                       ; BitmapPtr-Adressen errechnen
 move.l displaybase,a0                ; Basis des Display
 add.l #((breite/8)*höhe*tiefe),a0    ; plus einer Screengröße
 move.l #bitmap1,a1                   ; Bitmap1-Adresse
 bsr search_ptr                       ; BitmapPtr-Adressen errechnen

;---------- Bitmap-Struktur init. 0/1 ----------

 move.l gfxbase,a6                    ; gfxbase
 move.l #bitmap0,a0                   ; Bitmap0-Struktur
 move.l #tiefe,d0                     ; Depth
 move.l #breite,d1                    ; width
 move.l #höhe,d2                      ; heigth
 jsr -390(a6)                         ; InitBitMap ()
 move.l gfxbase,a6                    ; gfxbasis
 move.l #bitmap1,a0                   ; Bitmap1-Struktur
 move.l #tiefe,d0                     ; Depth
 move.l #breite,d1                    ; Width
```

```
 move.l #höhe,d2                     ; Heigth
 jsr -390(a6)                        ; InitBitMap ()

;---------- Screen öffnen ----------

 move.l intbase,a6                   ; intbase
 move.l #newscreen,a0                ; OSArgs-Struktur
 jsr -198(a6)                        ; OpenScreen ()
 move.l d0,screenhd                  ; Screen-Struktur speichern
 tst.l d0                            ; Error ?
 beq exit                            ; wenn ja, dann Ende
 move.l d0,a0                        ; Screenbasis nach A0
 add.l #44,a0                        ; plus 44
 move.l a0,viewport                  ; ViewPort-Adresse speichern
 move.l d0,a0                        ; Screenbasis nach A0
 add.l #84,a0                        ; plus 84
 move.l a0,rastport                  ; RastPort-Adresse speichern

 bsr initbob                         ; GelSystem initialisieren
 bsr setcollision                    ; Collisionsroutine setzen
 bsr changebitmaps                   ; Bitmaps vertauschen
 bsr printtext                       ; Text ausgeben in Bitmap1
 bsr changebitmaps                   ; BitMaps vertauschen
 bsr printtext                       ; Text ausgeben auch in Bitmap1
 bsr sortglist                       ; Gel-Liste sortieren
 bsr drawglist                       ; und ausgeben (Start-Positionen)

main:
 btst #10,$dff016                    ; rechte Maustaste ?
 beq exit
 bsr changebitmaps                   ; Bitmaps vertauschen
 bsr move_bob                        ; BOB bewegen
 bsr sortglist                       ; Gel-Liste sortieren
 move.l rastport,a1                  ; RastPort-Struktur
 move.l gfxbase,a6                   ; gfxbase
 jsr -108(a6)                        ; DoCollision ()
```

```
 bsr drawglist                          ; und ausgeben
 bsr no_flackern                        ; auf Austastlücke warten
 move.l intbase,a6
 jsr -384(a6)                           ; RemakeDisplay ()
 bra main

exit:
 move.l screenhd.a0                     ; wurde ein
 cmp.l #0,a0                            ; Screen geöffnet ?
 beq exit1                              ; wenn nicht, dann weiter
 move.l intbase,a6                      ; sonst Screen schließen
 jsr -66(a6)                            ; CloseScreen ()

exit1:
 move.l displaybase.a1                  ; wurde Speicher
 cmp.l #0,a1                            ; für Bitmaps reserviert ?
 beq exit2                              ; wenn nicht, dann weiter
 move.l #((breite/8)*tiefe*höhe*2),d0            ; Size
 move.l 4,a6                            ; execbasis
 jsr -210(a6)                           ; FreeMem ()

exit2:
 bsr closelibrarys                      ; Librarys schließen
 rts                                    ; Ende

;---------- Librarys öffnen ----------

openlibrarys:
 move.l 4,a6
 move.l #gfxname,a1                     ; Graphics
 clr.l d0
 jsr -552(a6)                           ; OpenLibrary ()
 move.l d0,gfxbase
 move.l 4,a6
 move.l #intname,a1                     ; Intuition
 clr.l d0
```

```
 jsr -552(a6)                         ; OpenLibrary ()
 move.l d0,intbase
 rts

;---------- Librarys schließen ----------

closelibrarys:
 move.l 4,a6
 move.l gfxbase,a1
 jsr -414(a6)                         ; CloseLibary ()
 move.l 4,a6
 move.l intbase,a1
 jsr -414(a6)                         ; CloseLibary ()
 rts

;---------- Collisonsroutine setzen ----------

setcollision:
 move.w #1,d0                         ; Routine 1 (MeMask/HitMask = 2)
 move.l #collisions_routine,a0        ; Routine mit RTS
 move.l #gelsinfo,a1                  ; GelsInfo-Struktur
 move.l gfxbase,a6                    ; gfxbasis
 jsr -144(a6)                         ; SetCollision ()

 move.w #2,d0                         ; Routine 2 (MeMask/HitMask = 4)
 move.l #collisions_routine,a0        ; Routine mit RTS
 move.l #gelsinfo,a1                  ; GelsInfo-Struktur
 move.l gfxbase,a6                    ; gfxbasis
 jsr -144(a6)                         ; SetCollision ()
 rts
```

```
;---------- Collisionsroutine welche bei Collision angesprungen wird
;---------- A1 = Oberer VSprite , A2 = Unterer VSprite -------------

collisions_routine:
 move.w #4095,d0                    ; Farb-Zähler laden
coll_loop:
 move.w d0,$dff180                  ; Hintergrundfarbe eintragen
 dbra d0,coll_loop                  ; Farb-Zähler um eins erniedrigen
 rts

;---------- Einen Text printen, damit man was sieht ----------

printtext:
 move.l rastport,a0                 ; RastPort-Struktur
 move.l #text1,a1                   ; Text-Struktur
 move.l #00,d0
 move.l #50,d1
 move.l intbase,a6
 jsr -216(a6)                       ; PrintIText ()
 rts

;---------- 100%iges No Flackern der Bobs ----------

no_flackern:
 move.l gfxbase,a6
 jsr -270(a6)                       ; WaitTOF
 rts

;---------- Gelsystem init. ----------

initbob:
 move.l #dummy1,a0                  ; Dummy1
 move.l #dummy2,a1                  ; Dummy2
 move.l #gelsinfo,a2                ; GelsInfo-Struktur
 move.l gfxbase,a6                  ; gfxbasis
 jsr -120(a6)                       ; InitGels ()
```

```
 move.l rastport,a0
 move.l #gelsinfo.20(a0)              ; Gelsystem in Rastport

 move.l #vsprite1,a0                  ; VSprite-Struktur
 move.l gfxbase,a6                    ; gfxbasis
 jsr -126(a6)                         ; InitMasks () von VSprite1

 move.l #vsprite2,a0                  ; VSprite-Struktur
 move.l gfxbase,a6                    ; gfxbasis
 jsr -126(a6)                         ; InitMasks () von VSprite2

 move.l #animob1,a0                   ; AnimOB-Struktur
 move.l #headob,a1                    ; Zeiger auf letztes AnimOB
 move.l rastport,a2                   ; RastPort-Struktur
 move.l gfxbase,a6                    ; gfxbasis
 jsr -156(a6)                         ; AddAnimOb () vom Objekt 1
 move.l #animob1,headob               ; Wichtig - letztes AnimOb

 move.l #animob2,a0                   ; AnimOB-Struktur
 move.l #headob,a1                    ; Zeiger auf letztes AnimOB
 move.l rastport,a2                   ; RastPort-Struktur
 move.l gfxbase,a6                    ; gfxbasis
 jsr -156(a6)                         ; AddAnimOb () vom Objekt 1
 rts

;---------- Bob bewegen ----------

move_bob:
 move.l #animob1.a0                   ; AnimOb-Strukur von Objekt 1
 move.w 18(a0),d0                     ; AnX-Position
 cmp.w #320*64.d0                     ; schon bei 320 ?
 bpl aneg                             ; wenn ja, dann Richtungswechsel
 cmp.w #-10*64,d0                     ; schon bei -10 ?
 bpl noneg                            ; wenn nicht, dann weiter
aneg:
 neg.w 22(a0)                         ; XVel = Vorzeichen wechseln
```

```
noneg:
 move.l #animob2,a0                 ; AnimOb-Strukur von Objekt 2
 move.w 18(a0),d0                   ; AnX-Position
 cmp.w #320*64,d0                   ; schon bei 320 ?
 bpl aneg2                          ; wenn ja, dann Richtungswechsel
 cmp.w #-10*64,d0                   ; schon bei -10 ?
 bpl noneg2                         ; wenn nicht, dann weiter

aneg2:
 neg.w 22(a0)                       ; XVel = Vorzeichen wechseln
noneg2:
 move.l #headob,a0                  ; Zeiger auf letztes AnimOb
 move.l rastport,a1                 ; RastPort-Struktur
 move.l gfxbase,a6                  ; gfxbasis
 jsr -162(a6)                       ; Animate ()
 rts

;---------- Displays im Rasterport vertauschen ----------

changebitmaps:
 move.l rastport,a1
 move.l viewport,a0
 move.l 36(a0),a0                   ; Zeiger auf RasInfo
 cmp.l #bitmap0,4(a0)
 bne no
 move.l #bitmap1,4(a0)
 move.l #bitmap1,4(a1)
 rts

no:
 move.l #bitmap0,4(a0)
 move.l #bitmap0,4(a1)
 rts
```

```
;---------- Bobs anzeigen ----------

sortglist:
 move.l rastport,a1              ; RastPort-Struktur
 move.l gfxbase.a6               ; gfxbasis
 jsr -150(a6)                    ; SortGList ()
 rts

drawglist:
 move.l rastport,a1              ; RastPort-Struktur
 move.l viewport,a0              ; ViewPort-Struktur
 move.l gfxbase,a6               ; gfxbasis
 jsr -114(a6)                    ; DrawGList ()
 rts

;---------- Bitmappointer errechnen ----------

                        ; A0 = Zeiger auf freien Speicher
                        ; A1 = Zeiger auf Bitmapxstruk.
search_ptr:
 move.l #breite/8,d0
 mulu #höhe,d0                   ; In D0 = Größe einer Bitmap
 move.w #tiefe-1,d1              ; In D1 = Anzahl Bitmaps
 move.l #8,d2

Ptr_loop:
 move.l a0,(a1,d2)               ; Ptr. auf Bitmaps in Bit-
                                 ; map0struk. saven
 add.l d0,a0                     ; eine Map weiter gehen
 add.l #4,d2                     ; Pointer-Offset plus 4
 dbra d1,Ptr_loop                ; Schleife bis alle Pointer durch
 rts
```

```
;---------- Parameter ----------

gfxname: dc.b "graphics.library",0
 even
intname: dc.b "intuition.library",0
 even
gfxbase: dc.l 0                    ; Graphics-Basis
intbase: dc.l 0                    ; Intuition-Basis

screenhd: dc.l 0

newscreen:
 dc.w 0,0,breite,höhe,tiefe
 dc.b 1,0
 dc.w 0,$140                       ; CustomBitMap (Wichtig)
 dc.l 0,0,0,bitmap0

viewport: dc.l 0                   ; Zeiger auf ViewPort-Struktur
rastport: dc.l 0                   ; Zeiger auf RastPort-Struktur
bitmap0: blk.b 40,0                ; Bitmap0-Struktur
bitmap1: blk.b 40,0                ; Bitmap1-Struktur

displaybase: dc.l 0
text1:
 dc.b 1,0,0,0
 dc.w 0,0
 dc.l 0,title,0
title: dc.b "DoubleBuffer BOB (rechte Maustaste)",0
 even

headob: dc.l 0                     ; Adresse des zuletzt eingefügten
                                   ; AnimOb
dummy1: blk.b 58,0
dummy2: blk.b 58,0
```

```
gelsinfo:                          ; ab hier GelsInfo-Struktur
 dc.b 0.0
 dc.l 0.0
 dc.l nextline
 dc.l lastcolor
 dc.l collhandler
 dc.w 0,0,0,0
 dc.l 0,0
nextline: blk.w 8.0
lastcolor: blk.w 32.0
collhandler: blk.l 16,0

;---------- ab hier Objekt 1 ----------

animob1:                ; ab hier AnimOb-Struktur
 dc.l 0,0               ; NextOb/PrevOb
 dc.l 0                 ; Clock
 dc.w 0.0               ; AnOldY/AnOldX
 dc.w 64*40.64*20       ; AnY,AnX - Koordinaten vom Objekt
 dc.w 0.45              ; YVel,XVel - Geschwindigkeit
 dc.w 0.0               ; YAccel,XAccel - keine Beschleunigung
 dc.w 0.0               ; RingYTrans,RingXTrans
 dc.l 0                 ; keine AnimORoutine
 dc.l animcomp1         ; HeadComp - Zeiger auf AnimComp-Struktur

animcomp1:              ; ab hier AnimComp-Struktur
 dc.w 0                 ; CompFlags = keine RingSequence
 dc.w 0,0               ; Timer,TimeSet = Null
 dc.l 0,0               ; NextComp,PrevComp
 dc.l 0.0               ; NextSeq,PrevSeq
 dc.l 0                 ; AnimCRoutine = keine
 dc.w 0,0               ; Y-Trans,X-Trans
 dc.l animob1           ; Zeiger auf dazugehörige AnimOb-Struktur
 dc.l bob1              ; Zeiger auf dazugehörige BOB-Struktur
```

```
bob1:                     ; ab hier BOB-Struktur
 dc.w 2                   ; Flag: BOBisComp
 dc.l savebuffer1         ; Puffer für Hintergrundspeicherung
 dc.l imageshadow1        ; Maskenpuffer
 dc.l 0.bob2              ; Priorität
 dc.l vsprite1            ; Zeiger auf dazugehörige VSprite-Struktur
 dc.l animcomp1           ; Zeiger auf dazugehörige AnimComp-Struktur
 dc.l dbuffer1            ; Zeiger auf DBufPacket für double-boufering

vsprite1:                 ; ab hier VSprite-Struktur
 dc.l 0,0                 ; Next/Prev-VSprite
 dc.l 0,0                 ; Draw/Clear-Path
 dc.w 0.0                 ; Old Y.X
 dc.w 6                   ; Flag: SaveBack/Overlay
 dc.w 0,0                 ; Y.X-Position wird in AnimOb-Struktur ge-
                          ; setzt
 dc.w 16.1                ; Höhe = 16 und Breite = 1 Word
 dc.w 2                   ; Tiefe = 2 BitMaps
 dc.w 2,4                 ; MeMask (Bit 1) / HitMask (Bit 2)
 dc.l imagedata1          ; Grafikdaten
 dc.l borderline1         ; Borderline
 dc.l imageshadow1        ; CollMask = ImageShadow
 dc.l 0                   ; SprColors
 dc.l bob1                ; Zeiger auf BOB-Struktur
 dc.b 3,0                 ; PlanePick/PlaneOnOff

dbuffer1:                 ; ab hier DBufPacket-Struktur
 dc.w 0.0
 dc.l 0
 dc.l bufbuffer1          ; Puffer zum Speichern des zweiten Hinter-
                          ; grundes
 dc.l 0
```

```
imagedata1:
 dc.w 0,0,0,0,$c60,$c60,$c60,$600c,$600c,$3018,$3018,$1c70,$fe0,$380
 dc.w 0,0,$7c0,$1ff0,$3ff8,$7ffc,$7ffc,$fffe,$fffe,$fffe,$fffe,$fffe
 dc.w $7ffc,$7ffc,$3ff8,$1ff0,$7c0,$0

savebuffer1: blk.b 16*1*2*5,0
bufbuffer1: blk.b 16*1*2*5,0          ; genauso groß wie SaveBuffer
imageshadow1: blk.b 16*1*2,0
borderline1: blk.b 1*2,0              ; Size = Bobbreite*2

;---------- ab hier Objekt 2 ----------

animob2:                      ; ab hier AnimOb-Struktur
 dc.l 0,0                     ; NextOb/PrevOb
 dc.l 0                       ; Clock
 dc.w 0,0                     ; AnOldY/AnOldX
 dc.w 64*40,64*230            ; AnY,AnX - Koordinaten vom Objekt
 dc.w 0,-65                   ; YVel,XVel - Geschwindigkeit
 dc.w 0,0                     ; YAccel,XAccel - keine Beschleunigung
 dc.w 0,0                     ; RingYTrans,RingXTrans
 dc.l 0                       ; keine AnimORoutine
 dc.l animcomp2               ; HeadComp - Zeiger auf AnimComp-Struktur

animcomp2:                    ; ab hier AnimComp-Struktur
 dc.w 0                       ; CompFlags = keine RingSequence
 dc.w 0,0                     ; Timer,TimeSet = Null
 dc.l 0,0                     ; NextComp,PrevComp
 dc.l 0,0                     ; NextSeq,PrevSeq
 dc.l 0                       ; AnimCRoutine = keine
 dc.w 0,0                     ; Y-Trans,X-Trans
 dc.l animob2                 ; Zeiger auf dazugeh÷rige AnimOb-Struktur
 dc.l bob2                    ; Zeiger auf dazugeh÷rige BOB-Struktur
```

```
bob2:                        ; ab hier BOB-Struktur
 dc.w 2                      ; Flag: BOBisComp
 dc.l savebuffer2            ; Puffer für Hintergrundspeicherung
 dc.l imageshadow2           ; Maskenpuffer
 dc.l bob1,0                 ; Priorität
 dc.l vsprite2               ; Zeiger auf dazugehörige VSprite-Struktur
 dc.l animcomp2              ; Zeiger auf dazugehörige AnimComp-Struktur
 dc.l dbuffer2               ; Zeiger auf DBufPacket für double-boufering

vsprite2:                    ; ab hier VSprite-Struktur
 dc.l 0,0                    ; Next/Prev-VSprite
 dc.l 0,0                    ; Draw/Clear-Path
 dc.w 0,0                    ; Old Y,X
 dc.w 6                      ; Flag: SaveBack/Overlay
 dc.w 0,0                    ; Y,X-Position wird in AnimOb-Struktur ge-
                             ; setzt
 dc.w 16,1                   ; Höhe = 16 und Breite = 1 Word
 dc.w 2                      ; Tiefe = 2 BitMaps
 dc.w 4,2                    ; MeMask (Bit 2) / HitMask (Bit 1)
 dc.l imagedata2             ; Grafikdaten
 dc.l borderline2            ; Borderline
 dc.l imageshadow2           ; CollMask = ImageShadow
 dc.l 0                      ; SprColors
 dc.l bob2                   ; Zeiger auf BOB-Struktur
 dc.b 2,1                    ; PlanePick/PlaneOnOff

dbuffer2:                    ; ab hier DBufPacket-Struktur
 dc.w 0,0
 dc.l 0
 dc.l bufbuffer2             ; Puffer zum Speichern des zweiten Hinter-
                             ; grundes
 dc.l 0
```

```
imagedata2:
 dc.w 0,0,0,0,$c60,$c60,$c60,$600c,$600c,$3018,$3018,$1c70,$fe0,$380
 dc.w 0,0,$7c0,$1ff0,$3ff8,$7ffc,$7ffc,$fffe,$fffe,$fffe,$fffe,$fffe
 dc.w $7ffc,$7ffc,$3ff8,$1ff0,$7c0,$0

savebuffer2: blk.b 16*1*2*5,0
bufbuffer2: blk.b 16*1*2*5,0          ; genauso groß wie SaveBuffer
imageshadow2: blk.b 16*1*2,0
borderline2: blk.b 1*2,0              ; Size = Bobbreite*2
END
;---------------------------- Listingende -------------------------
```

8.9 Tips & Tricks zu Grafikobjekten

Mit dem "double-buffering" kann man zwar das Flackern abstellen, jedoch ruckeln einige Objekte dabei. Darum folgen hier ein paar Tips, wie man das Ruckeln weitgehenst ausschalten kann.

Wie es schon im letzten Listing angewendet wurde, sollte man die Objekte nicht über ein eigenes Display sondern über einen mit der Routine "OpenScreen ()" erzeugten Screen bewegen. Denn so erspart man sich das umständliche Neuberechnen der Copperlisten. Bei Screens braucht lediglich die Routine "RemakeDisplay ()" aufgerufen werden. Dabei wird kurzzeitig das Multitasking abgeschaltet. Dadurch kann das eigene Programm nicht mehr von anderen gestört werden.

Eine weitere Möglichkeit ist, das Multitasking für die ganze Zeit auszuschalten. Dieses erledigt die Exec-Routine "Forbid ()". Sie benötigt keine Paramter und schaltet das gesammte Multitasking ab. Mit der Exec-Routine "Permit ()" kann das Multitasking wieder eingeschaltet werden.

Oder man schaltet den Workbench-Screen ganz am Anfang seines Programmes aus. Es werden dann zwar keine Mausfunktionen mehr unterstützt, dafür steht einem aber

1. mehr Speicher zur Verfügung und
2. das Ruckeln der Objekte ist sehr gering.

Den Workbench-Screen kann man mit der Intuition-Routine "CloseWorkbench ()" schließen.

9. Kapitel

Iff Standart

- Iff Standart
- Screens
- Brushes (BOBs)

9. Iff Standart

Stellen sie sich mal vor, sie haben ein Bild mit Deluxe Paint erstellt und speichern es auf Diskette ab. Sie haben gerade ein kleines Programm geschrieben, worin sie dieses Bild als Titelbild einbauen wollen.

Woher wissen sie jetzt, wie breit, hoch, oder lang das Bild ist, welche Farben verwendet wurden, wann die eigentlichen Bilddaten anfangen oder wie es entpacked wird usw..
Dafür wurde von dem Softwarehaus Electronic Arts der "Interchange File Format" Standart, abgekürzt "IFF", entwickelt. Er steht für den Datenaustausch zwischen verschiedenen Programmen.

Das heißt, das ein Bild, was mit Deluxe Paint gemalt wurde auch von einem anderem Malprogramm wieder geladen und somit bearbeitet werden kann.

Es gibt den Iff-Standart auch in Sachen Musik, Text usw.. Wir wollen hier allerdings nur auf den Iff-Standart für Graphik eingehen.

Das Verfahren ist ziemlich einfach nach dem die Bilder auf Diskette abgespeichert werden. Es wird einfach ein Header vor den eigenlichen Bilddaten abgespeichert. In diesem Header sind alle nötigen Informationen enthalten, die für die Darstellung eines Bildes notwendig sind.

Der Header selbst wird in kleinen Blöcken aufgeteilt, die als "Chunks" bezeichnent werden. Jeder "Chunk" (Block) beginnt mit einem Namen, danach folgt die Länge des Chunks und dann die eigentlichen Informationen. Durch diese Methode ist der Iff-Standart jeder Zeit Erweiterungen offen, denn man kann beliebig viele Chunks hinzufügen.

9.1 Screens

Um nun ein IFF-Bild anzeigen zu können, fehlt uns noch das wissen, wie der Header nun aufgebaut ist. Wie schon erwähnt ist der Header in mehrere Chunks (Blöcke) aufgeteilt. Wir werden hier nur auf die wichtigsten Blöcke eingehen, die nötig sind um ein IFF-Bild anzeigen zu können.

1. FORM-Chunk-Struktur (Länge = 12 Bytes)

Offset	Typ	Inhalt
000	Long	FORM
004	Long	Gesamtlänge des Bildes
008	Long	ILBM

2. BMHD-Chunk-Struktur (Länge = 28 Bytes)

Offsct	Typ	Inhalt
000	Long	BMHD
004	Long	Gesamtlänge des Block (20 Bytes)
008	Word	Breite der Grafik
010	Word	Höhe der Grafik
012	Word	X-Position der Grafik
014	Word	Y-Position der Grafik
016	Byte	Anzahl der Bitplanes
017	Byte	Masking 0 = kein Masking 1 = Masking 2 = Transparent 3 = Lasso
018	Byte	DatenKompression 0 = nicht gepacked 1 = gepacked

019	Byte	unbenutzt
020	Word	Transparentfarbe
022	Byte	X-Aspekt
023	Byte	Y-Aspekt
024	Word	Breite der Quellseite
026	Word	Höhe der Quellseite

3. CMAP-Chunk-Struktur (Länge = 104 Bytes)

Offsct	Typ	Inhalt
000	Long	CMAP
004	Long	Gesamtlänge des Blocks (96 Bytes)
008	Byte	Farbe0 Rotanteil
009	Byte	Farbe0 Grünanteil
010	Byte	Farbe0 Blauanteil

usw .... für alle anderen Farbregister bis Farbregister 31

4. BODY-Chunk-Struktur (Länge = abhängig vom Bild)

Offset	Typ	Inhalt
000	Long	BODY
004	Long	Gesamtlänge der Bilddatcn
008	Byte	Datenbytes

.... hier folgen die eigentlichen Datenbytes

Da die meisten Bilder aus Platzgründen gepacked auf Diskette abgcspcichcrt wcrdcn, folgt dic Bcschrcibung dcs Packvorgangs:

Ob ein Bild gepacked ist oder nicht, crkcnnt man an dcm Compressions-Byte im BMHD-Chunk. Ist es eins, wurde das Bild gepacked auf Diskctte gespeichert.

Ein Iff-Bild wird nicht Bitplaneweise gespeichert, wie sie vielleicht denken, sondern Zeilenweise. Das bedeutet, es kommt zuerst die erste Zeile von Bitplane 1, dann die erste Zeile von Bitplane 2 und so weiter bis zur letzten. Dann folgt die ersten Zeile der Maskenplane, allerdings nur wenn das Masking-Byte nicht null ist. Nun folgt die zweite Zeile von Bitplane 1, dann die zweite Zeile von Bitplane 2 usw.

Wie funktioniert nun der Packvorgang?

Es darf immer nur eine Zeile für sich entpackt werden, niemals zwei auf einmal; also Zeilenweise entpacken.

Die gepackten Daten sind wie folgt Strukturiert:

Erst kommt das Befehlsbyte für den Entpacker, welches wir als "n" bezeichnen. Ist "n" eine Zahl zwischen 0 und 127, so werden die nächsten (n + 1) Bytes unverändert übernommen. Ist "n" jedoch negativ, also eine Zahl zwischen -1 bis -127, so wird das folgende Byte (-n+1) mal wiederholt. Der Wert -128 hat keine Funktion und kann übersprungen werden. Das ist eigenlich schon alles, was zum Packvorgang zu sagen ist. Wie sie sehen, so kompliziert ist er eigenlich gar nicht.

Das folgende Beispielprogramm lädt ein IFF-Bild in den Speicher, entpacked es und zeigt es auf dem Bildschirm an. Mit der rechten Maustaste verschwindet das Bild wieder aus dem Speicher. Durch ändern des Filenames, kann man jedes beliebige IFF-Bild laden. Der Filename befindet sich in den Paramtern am Ende des Listings. Das Programm muß sich auf der selben Diskette befinden, wie das Bild, das angezeigt werden soll.

```
;---------------------- Programmname = Iff-Loader ------------------
;----------------------- rechte Maustaste = Ende --------------------
;------------------ lädt ein Iff-Pic in den Speicher ----------------
;
start:
;---------- Librarys öffnen ----------

 clr.l d0                           ; Version = Null
 move.l 4,a6                        ; execbase
 move.l #gfxname,a1                 ; graphics-library
 jsr -552(a6)                       ; OpenLibrary ()
 move.l d0,gfxbase                  ; Basis speichern

 move.l 4,a6                        ; Execbase
 clr.l d0                           ; Version = Null
 move.l #intname,a1                 ; Intuition-Library
 jsr -552(a6)                       ; OpenLibrary ()
 move.l d0,intbase                  ; Basis speichern

 move.l 4,a6                        ; Execbasis
 clr.l d0                           ; Version
 move.l #dosname,a1                 ; Dos-Library
 jsr -552(a6)                       ; OpenLibrary ()
 move.l d0,dosbase                  ; Basis speichern


;---------- File öffnen ----------

 move.l dosbase,a6                  ; Dos-Basis
 move.l #filename,d1                ; Filename
 move.l #1005,d2                    ; Modus = Old
 jsr -30(a6)                        ; OPEN ()
 tst.l d0                           ; Error ?
 beq prg_end                        ; Wenn ja, dann Prg.ende
 move.l d0,bif                      ; Sonst Filehd saven
```

```
;---------- ersten ( Bytes lesen ) ----------

 move.l #disk_buf,d2           ; Puffer für ersten 8 Bytes
 move.l #8,d3                  ; Länge = 8 Bytes
 move.l dosbase,a6             ; Dos-Basis
 move.l bif,d1                 ; Filehd
 jsr -42(a6)                   ; READ ()

 move.l #disk_buf,a0           ; Puffer der ersten 8 Bytes
 move.l 4(a0),d3               ; Länge des Files nach D3
 move.l d3,chunk_size          ; und speichern

;---------- Speicher für Picture reservieren ----------

 move.l 4,a6                   ; Execbasis
 move.l #$10002,d1             ; Chip + FreeMem
 move.l chunk_size,d0          ; Länge nach D0
 jsr -198(a6)                  ; AllocMem ()
 move.l d0,pic_buf             ; Adresse speichern
 tst.l d0                      ; Error ?
 beq prg_end                   ; wenn ja, dann Prg.ende

;---------- Picture in Speicher laden ----------

 move.l chunk_size,d3          ; Länge nach D3
 move.l pic_buf,d2             ; Pufferadresse nach D2
 move.l dosbase,a6             ; Dos-Basis
 move.l bif,d1                 ; Filehd
 jsr -42(a6)                   ; READ ()

;---------- File wieder schließen ----------

 move.l dosbase,a6             ; Execbasis
 move.l bif,d1                 ; Filehd
 jsr -36(a6)                   ; CLOSE ()
```

```
;---------- Adressen der Iff-Chunks suchen ----------

 jsr iff_search

;---------- Screen öffnen für Iff-Pic ----------

 move.l intbase,a6               ; Intbasis
 move.l #newscreen,a0            ; OSArgs
 jsr -198(a6)                    ; OpenScreen ()
 move.l d0,screen                ; Screenhd speichern

 jsr planes_init                 ; PlanePtr-Adressen speichern
 jsr colours_init                ; Farben herstellen
 jsr black_colour_set            ; Screen schwarz einfärben

;---------- Parameter für Entpacker ----------

 move.l #planes,bitplanezeiger   ; Puffer von PlanePtr-Adressen
 move.l bmhd_chunk,a0            ; BMHD-Chunk-Adresse
 move.b 17(a0),maske             ; MaskenByte setzen
 move.b 18(a0),compression       ; PackerByte
 clr.w d0
 move.b 16(a0),d0                ; Depth
 move.w d0,anzahlplanes          ; speichern
 move.w 8(a0),picbreite          ; Picbreite
 move.l body_chunk,a0            ; BODY-Chunk-Adresse
 add.l #4,a0                     ; plus 4
 move.l (a0)+,piclänge           ; = Piclänge
 move.l a0,picadresse            ; A0 = Pic-Anfangsadresse
 jsr unpacker                    ; entpacken
 jsr colour_set                  ; Iff Farben setzen
```

```
;---------- Speicher von Iff-Daten freigeben ----------

 move.l 4,a6                          ; Execbasis
 move.l pic_buf,a1                    ; Picadresse
 move.l chunk_size,d0                 ; Länge
 jsr -210(a6)                         ; FreeMEM ()

;---------- Auf rechte Maustaste warten ----------

wait_loop:
 btst #10,$dff016
 bne wait_loop

;---------- Programmende ----------

prg_end:
 move.l screen,a0                     ; Screen
 cmp.l #0,a0                          ; vorhanden ?
 beq prg_end1                         ; wenn nicht, dann weiter
 move.l intbase,a6                    ; sonst
 jsr -66(a6)                          ; CLOSE ()

;---------- Librarys schließen ----------

prg_end1:
 move.l 4,a6
 move.l intbase,a1
 jsr -414(a6)                         ; CloseLibrary ()
 move.l 4,a6
 move.l gfxbase,a1
 jsr -414(a6)                         ; CloseLibarry ()
 move.l 4,a6
 move.l dosbase,a1
 jsr -414(a6)                         ; CloseLibrary ()
 rts
```

```
;---------- Iff-Unpacker-Routine ----------

unpacker:
 clr.l d0
 move.w picbreite.d0              ; Picbreite nach D0
 lsr.w #3.d0                      ; durch 8 teilen
 move.w d0,width_bytes            ; und speichern
 move.w anzahlplanes,d2           ; Depth nach D2
 move.l bitplanezeiger,a2         ; Pufferadresse v. PlanePtr
 move.l picadresse.a0             ; Picadresse nach A0
 move.l a0.a3                     ; nach A3
 add.l piclänge,a3                ; plus Piclänge = Picende

unp_loop:
 cmp.l a3,a0                      ; Bild schon entpacked ?
 bge unpack_end                   ; wenn ja, dann Ende
 clr.w d3                         ; BitMapzähler

pic_loop1:
 move.w d3,d4                     ; D3 nach D4
 lsl.w #2.d4                      ; mal 4
 move.l (a2.d4),a5                ; Zeiger auf Bitmap errechnen
 jsr unpack_row                   ; und Daten entpacken
 move.l a5,(a2.d4)                ; BitMapposition speichern
 addq.w #1,d3                     ; BitMapzähler plus 1
 cmp.w d2.d3                      ; schon eine Reihe aller Maps
                                  ; durch ?
 blt pic_loop1                    ; wenn nicht. dann weiter
 andi.b #1,maske                  ; sonst Maske entpacken
 beq unp_loop                     ; wenn vorhanden ?
 move.l #mask_dummy.a5            ; Maskenpuffer
 jsr unpack_row                   ; entpacken
 bra unp_loop                     ; wieder von vorne
```

```
unpack_row:
 move.l d2,-(sp)                      ; Depth-Variable retten
 move.w width_bytes,d2                ; Zeilenbreite in Bytes

unp_loop1:
 tst.w d2                             ; schon eine Zeile durch ?
 beq unpack_row_end                   ; Wenn ja, dann nächste
 clr.w d0
 tst.b compression                    ; Pic gepacked ?
 bne unp_comp                         ; wenn ja, dann Unp_Comp
 move.w width_bytes,d0                ; sonst Zeilenbreite nach D0
 subq.w #1,d0                         ; minus 1
 bra unp_loop2                        ; und Bytes übernehmen

unp_comp:
 move.b (a0)+,d0                      ; Befehlsbyte holen
 bmi packed                           ; Wenn negativ, dann gepacked

unp_loop2:
 move.b (a0)+,(a5)+                   ; sonst Bytes normal ⁿbernehmen
 subq.w #1,d2                         ; Zeilenbreite minus 1
 dbra d0,unp_loop2                    ; Schleife bis Anzahl Null
 bra unp_loop1                        ; Von vorne

packed:
 neg.b d0                             ; Befehlsbyte negieren
 move.b (a0)+,d1                      ; Füllbyte nach d1

unp_loop3:
 move.b d1,(a5)+                      ; Zeile mit
 subq.w #1,d2                         ; Füllbyte beschreiben
 dbra d0,unp_loop3                    ; Schleife
 bra unp_loop1                        ; vor vorne

unpack_row_end:
 move.l (sp)+,d2                      ; Depth-Variable wiederholen
```

```
unpack_end:
 rts                                ; Ende

;---------- Parameter für Entpacker ----------

maske: dc.w 0
compression: dc.w 0
anzahlplanes: dc.w 0
bitplanezeiger: dc.l 0
picbreite: dc.w 0
picl
änge: dc.l 0
picadresse: dc.l 0

;---------- PlanePointer-Adressen errechnen ----------

planes_init:
 move.l screen,a0
 add.l #$c0,a0
 move.l #planes,a1
 move.w planes_num.d0

plane_loop:
 move.w d0,d1
 mulu #4,d1
 move.l (a0,d1),(a1,d1)
 dbra d0,plane_loop
 rts

;---------- Iff Farben errechnen und speichern ----------

colours_init:
 move.l cmap_chunk,a0
 add.l #8,a0
 move.l #colour_map,a1
 move.w #31,d7
```

```
co_loop1:
 clr.w d0
 move.b (a0)+,d0
 and.b #$f0,d0
 lsl.w #4,d0
 move.b (a0)+,d0
 and.w #$0ff0,d0
 clr.w d1
 move.b (a0)+,d1
 lsr.b #4,d1
 and.b #$0f,d1
 or.b d1,d0
 move.w d0,(a1)+
 dbra d7,co_loop1
 rts

;---------- Iff Farben setzen ----------

colour_set:
 move.l screen,a0
 add.l #44,a0
 move.l #colour_map,a1
 move.w #32,d0
 move.l gfxbase,a6
 jsr -192(a6)                       ; LoadRGB4 ()
 rts

;---------- Screen schwarz einfärben ----------
black_colour_set:
 move.l screen,a0
 add.l #44,a0
 move.l #black_colour,a1
 move.w #32,d0
 move.l gfxbase,a6
 jsr -192(a6)                       ; LoadRGB4 ()
 rts
```

```
;---------- Iff-Chunk-Adressen suchen ----------

iff_search:
 move.l #chunktabelle,a1
 move.l #chunkadresse,a2
iffsearch0:
 move.l pic_buf,a0
 clr.w d0

iffsearch1:
 cmp.l #0,(a1)
 beq iffsearch2
 move.b (a0)+,iffchunk
 move.b (a0)+,iffchunk+1
 move.b (a0)+,iffchunk+2
 move.b (a0)+,iffchunk+3
 move.l iffchunk,d6
 move.l (a1),d7
 cmp.l d7,d6
 beq iffsearch3
 sub.l #3,a0
 bra iffsearch1

iffsearch00:
 add.l #4,a1
 add.l #4,a2
 bra iffsearch0

iffsearch3:
 sub.l #4.a0
 move.l (a2),a4
 move.l a0,(a4)
 bra iffsearch00
```

```
iffsearch2:
 clr.w d0
 move.l bmhd_chunk,a0
 move.b 16(a0),d0
 move.w d0,newscreen+8
 sub.w #1,d0
 move.w d0,planes_num
 move.w 8(a0),width
 move.w 10(a0),height
 rts

;---------- Parameter für Programm ----------

chunktabelle: dc.l "BMHD","CMAP","BODY",0
chunkadresse: dc.l bmhd_chunk,cmap_chunk,body_chunk
iffchunk: dc.l 0

bmhd_chunk: dc.l 0
cmap_chunk: dc.l 0
body_chunk: dc.l 0

planes_num: blk.w 1.0
width_bytes: blk.w 1,0

dosname: dc.b "dos.library",0
 even
intname: dc.b "intuition.library",0
 even
gfxname: dc.b "graphics.library",0
 even
gfxbase: blk.l 1,0
intbase: blk.l 1,0
dosbase: blk.l 1,0

newscreen:
 dc.w 0,0
```

```
width: dc.w 320
height: dc.w 256
 dc.w 5
 dc.b 0,1
modes: dc.w 0,15
 dc.l 0
 dc.l 0
 dc.l 0,0

screen: dc.l 0
planes: blk.l 10.0
colour_map: blk.w 32,0
black_colour: blk.w 32,0

filename: dc.b "iff-pic",0
 even
bif: dc.l 0
disk_buf: blk.b 10,0
pic_buf: blk.l 1,0
chunk_size: blk.l 1,0
mask_dummy: blk.b 128,0

;------------------------- Ende des Listings ----------------------
```

9.2 Brushes (BOBs)

Jedes Malprogramm besitzt Fuktionen, meist unter dem Menüpunkt "Brush", mit denen man Teile aus einem Bild ausschneiden und diesen dann als Pinsel benutzten kann. Diese Pinsel werden dann als Brush bezeichnet und können beliebig groß und breit sein. Sie werden genau nach dem selben Verfahren abgespeichert, wie die Bilder. Jedoch sind sie meistens nicht gepacked und das entpacken ist auch etwas schwieriger. Da BOBs immer eine Breite die durch 16 teilbar ist besitzten müssen.

Da mir noch kein Buch oder eine Zeitschrift bekannt ist, in der ein Listing in Assembler veröffentlicht wurde, welches IFF-Brushes entpacked und alle nötigen Parameter dazu abspeichert, die dann leicht ausgewertet werden können, um einen BOB zu aktivieren, habe ich selbst eins geschrieben.

Die Größe des Brushes kann beliebig sein. Das Programm wandelt einen Brush in Bruchteilen von Sekunden um und speichert ihn dann auf Diskette.

Das Programm ist ziemlich primitiv gehalten worden, denn es soll ja bloß das Verfahren demonstrieren, nachdem Brushes entpacked werden. Wenn sie wollen, können sie ja durch erweitern des Programmes, daraus einen komfortablen Brush-Transformer programmieren (mit Menüs etc.).

Anleitung zum Programm:

Nach dem Programmnamen müssen sie als Parameter den Filename des Brush übergeben, welcher umgewandelt werden soll. Brush und Programm brauchen sich dafür allerdings nicht auf der selben Diskette zu befinden.

Beispiel: Brush-Transformer Hund

Danach werden sie aufgefordert die Diskette einzulegen, auf der sich der Brush befindet. Haben sie dies getan, drücken sie Return und der Brush wird in den Speicher geladen.

Jetzt wird der Brush umgewandelt. Nun werden sie aufgefordert eine Diskette einzulegen, auf die der umgewandelte Brush abgespeichert werden soll. Haben sie dies getan, drücken sie Return und es wird gespeichert.

Der umgewandelte Brush wird unter dem Namen "TransBOB" abgespeichert.

Aufbau des "TransBOB" auf Diskette:

Offset	Typ	Inhalt
000	Long	TBOB
004	Long	Gesamtlänge der Datei
008	Word	Höhe des BOBs in Zeilen
010	Word	Anzahl Words die der BOB breit ist.
012	Word	Depth (Anzahl Planes des BOBs)
014	Byte	PlanePick-Wert
015	Byte	PlaneOnOff-Wert
016	Byte	Datenbytes Bitplanes werden nacheinander abgespeichert

Die eigenlichen Grafikdaten des BOBs beginnen ab Offset 16. Erst folgen die Datenbytes von Bitplane 1, dann die von Bitplane 2 usw. Es folgen so viele Bitplanes, wie in Depth (Offset 12) angegeben wird. Die Grafikdaten liegen jetzt also Bitplaneweise vor, nicht mehr zeilenweise wie vor der Umwandlung.

```
;------------------ Programmname = Brush-Transformer ----------------
;--------------- wandelt ein Brush in eine BOB-Datei um -------------

;---------- Filename einlesen ----------

 sub.l #1,d0                            ; Textlänge -1
 tst.l d0                               ; liegt Text vor ?
 bne ok                                 ; wenn ja, dann weiter
 rts                                    ; sonst Ende

ok:
 move.l d0,textlänge                    ; Länge des Textes speichern
 move.l a0,textadresse                  ; Anfangsadresse des Textes
                                        ; speichern
 add.l d0,a0                            ; Textende
 move.b #0,(a0)                         ; Mit Nullbyte abschließen

;---------- Dos-Library öffnen ----------

 move.l 4,a6
 move.l #dosname,a1
 clr.l d0
 jsr -552(a6)                           ; OpenLibrary ()
 move.l d0,dosbase

;---------- auf Returntaste warten ----------

 move.l #text1,textanfang               ; Textadresse
 bsr meldung                            ; und ausgeben

get:
 cmp.b #$77,$bfec01
 bne get
```

```
get1:
 cmp.b #$76,$bfec01
 bne get1

;---------- IFF-Header in Speicher laden ----------

 move.l #1005,d2                    ; Mode = OLD
 move.l textadresse,d1              ; Filename
 move.l dosbase,a6
 jsr -30(a6)                        ; OPEN ()
 move.l d0,filehd                   ; Fileadresse speichern
 tst.l d0                           ; File auf Disk ?
 bne ok1                            ; wenn ja, dann ok1
 move.l #text2,textanfang           ; Textadresse
 bsr meldung                        ; und ausgeben (Error)
 bra exit                           ; Programmende

ok1:
 move.l filehd,d1                   ; Fileadresse
 move.l #puffer,d2                  ; Puffer für
 move.l #12,d3                      ; ersten 12 Bytes
 move.l dosbase,a6
 jsr -42(a6)                        ; READ ()
 move.l filehd,d1                   ; File wieder schließen
 move.l dosbase,a6
 jsr -36(a6)                        ; CLOSE ()

;---------- ab hier Diskheader auswerten ----------

 lea puffer,a0
 cmp.l #"FORM",(a0)                 ; liegt IFF-Datei vor ?
 beq ok2                            ; wenn ja, dann ok2
 move.l #text3,textanfang           ; textadresse
 bsr meldung                        ; und ausgeben (Error)
 bra exit                           ; Programmende
```

```
ok2:
 cmp.l #"ILBM",8(a0)              ; liegt ILBM-Format vor ?
 beq ok3                          ; wenn ja, dann ok3
 move.l #text4,textanfang         ; textadresse
 bsr meldung                      ; und ausgeben (Error)
 bra exit                         ; Programmende

;---------- Speicher für Brush reservieren ----------

ok3:
 move.l 4(a0),d0                  ; Dateigröße
 move.l #$10002,d1                ; Chip- und Free-Memory
 move.l 4,a6
 jsr -198(a6)                     ; AllocMem ()
 move.l d0,brush_base             ; Adresse speichern
 tst.l d0                         ; konnte Speicher reserviert
                                  ; werden ?
 bne ok4                          ; wenn ja, dann ok4
 move.l #text5,textanfang         ; sonst Erromeldung
 bsr meldung                      ; ausgeben
 bra exit                         ; Programmende

;---------- Brush in Speicher laden ----------

ok4:
 move.l #1005,d2                  ; Modus = Alt
 move.l textadresse,d1            ; Filename
 move.l dosbase,a6
 jsr -30(a6)                      ; OPEN ()
 move.l d0,filehd                 ; Fileadresse
 move.l d0,d1
 move.l brush_base,d2             ; Datenpuffer
 move.l puffer+4,d3               ; Datenlänge
 move.l dosbase,a6
 jsr -42(a6)                      ; READ ()
 move.l filehd,d1                 ; Fileadresse
```

```
 move.l dosbase.a6
 jsr -36(a6)                    ; CLOSE


;---------- IFF-Brush-Struktur auswerten ----------

 move.l #chunktabelle,a1        ; Brush-Chunkadressen suchen
 move.l #chunkadresse,a2        ; dort werden Adressen gespei-
                                ; chert

biffsearch0:
 move.l brush_base,a0           ; Brushanfang

biffsearch1:
 cmp.l #0,(a1)                  ; schon alle Chunks gefunden ?
 beq chunk_search_ende          ; wenn ja, dann Ende
 move.b (a0)+,iffchunk          ; sonst
 move.b (a0)+,iffchunk+1        ; erstes
 move.b (a0)+,iffchunk+2        ; LongWord
 move.b (a0)+,iffchunk+3        ; lesen
 move.l iffchunk,d6             ; LongWord nach D6
 move.l (a1),d7                 ; mit Chunk aus
 cmp.l d7,d6                    ; Tabelle vergleichen
 beq biffsearch3                ; wenn gleich, dann Adresse
                                ; speichern
 sub.l #3,a0                    ; sonst Brushadresse minus 3
 bra biffsearch1                ; und weiter suchen
biffsearch3:                    ; ab hier Chunkadresse speichern
 sub.l #4,a0                    ; Brushadresse minus 4
 move.l (a2),a4                 ; Puffer aus Tabelle holen
 move.l a0,(a4)                 ; und Chunkadresse speichern
 add.l #4,a1                    ; Chunktabelle plus 4
 add.l #4,a2                    ; Puffertabelle plus 4
 bra biffsearch0                ; nächsten Chunk aus Tabelle su-
                                ; chen

chunk_search_ende:
```

```
;---------- BMHD-Chunk auswerten für Unpacker ----------

 move.l bmhd_chunk,a0            ; BMHD-Chunkadresse nach A0
 move.b 18(a0),packerbyte        ; Compression-Byte speichern
 move.w 10(a0),bob_höhe          ; Bobhöhe speichern
 clr.l d0
 move.w 8(a0),d0                 ; BOB-Breite nach D0
 divu #16,d0                     ; durch 16 teilen
 swap d0                         ; HIGH- und LOW-Word vertauschen
 cmp.w #0,d0                     ; Rest der Division = Null ?
 bne daw1                        ; wenn nicht, dann daw1
 swap d0                         ; sonst, Words wieder vertauschen
 bra daw2                        ; und daw2

daw1:
 swap d0                         ; Words wieder vertauschen
 add.w #1,d0                     ; plus 1 Word (wegen Rest)

daw2:
 lsl.w #1,d0                     ; Wordbreite mal 2 = Bytebreite
 move.w d0,bob_bytebreite        ; Bobbreite (in Bytes) speichern
 clr.w d0
 move.b 16(a0),d0                ; Depth vom Brush
 move.w d0,bob_planes            ; Anzahl Bob-Planes speichern
;
 move.b #$ff,d7                  ; D7 mit 255 laden
 move.w #7,d1                    ; max. 8 Planes (7 wegen DBRA)
 sub.w d0,d1                     ; max. Planes minus Anzahl Planes

loop_pick:
 lsr.b #1,d7                     ; PlanePick-Wert ermitteln
 dbra d1,loop_pick
 move.b d7,planepick             ; PlanePick speichern
 move.b #0,planeonoff            ; PlaneOnOff = 0
```

```
;---------- Speicher fürs Entpacken reservieren -----------

 move.w bob_planes.d0              ; Anzahl Planes
 mulu bob_bytebreite,d0            ; mal BOB-Breite
 mulu bob_höhe,d0                  ; mal BOB-Höhe
 move.l d0,bob_planes_size         ; = BOB-Größe
 move.l #$10002,d1                 ; CHIP- und FREE-Memory
 move.l 4,a6
 jsr -198(a6)                      ; AllocMem ()
 move.l d0,bob_planes_base         ; Adresse speichern
 tst.l d0                          ; konnte Speicher reserviert
                                   ; werden ?
 bne bob_ok6                       ; wenn ja, dann ok6
 move.l #text5,textanfang          ; sonst Errormeldung
 bsr meldung                       ; ausgeben
 bra exit                          ; Programmende

;---------- Anfangsadresse der BitMaps errechnen und speichern -----

bob_ok6:
 move.l d0,a0                      ; Adresse der 1. Bitplane
 move.w bob_bytebreite,d0          ; BOB-Bytebreite nach D0
 mulu bob_höhe,d0                  ; mal BOB-Höhe = Bitplanegröße
 move.l #bitmapadressen,a1         ; Puffer für Bitplane-Pointer
 move.w bob_planes,d1              ; Anzahl Planes
 sub.w #1,d1                       ; minus 1, wegen DBRA-Befehl

buploop:
 move.l a0,(a1)+                   ; Bitplaneadresse speichern
 add.l d0,a0                       ; plus eine Bitplanegröße
 dbra d1,buploop                   ; weiter speichern
```

```
;---------- Brush entpacken ----------

 bsr start_dekompression
 move.l #text6,textanfang           ; Ok-Meldung
 bsr meldung                        ; ausgeben

;---------- BOB-Header aufbauen ----------

 lea bob_header,a0                  ; Headeranfangsadresse
 move.l #"TBOB",(a0)                ; Bezeichnung
 move.l bob_planes_size,4(a0)       ; BOB-ImageSize
 add.l #16,4(a0)                    ; plus BOB-Header
 move.w bob_höhe,8(a0)              ; Höhe
 move.w bob_bytebreite,d0           ; Breite in Bytes
 lsr.w #1,d0                        ; durch 2 = Breite in Words
 move.w d0,10(a0)                   ; und Wordbreite speichern
 move.w bob_planes.12(a0)           ; Tiefe
 move.b planepick.14(a0)            ; PlanePick
 move.b planeonoff,15(a0)           ; PlaneOnOff

;---------- Auf Disk warten worauf BOB gespeichert werden kann -----

 move.l #text7,textanfang           ; Text
 bsr meldung                        ; ausgeben

geta:
 cmp.b #$77,$bfec01
 bne geta

geta1:
 cmp.b #$76,$bfec01
 bne geta1
```

```
;---------- BOB-Header auf Disk schreiben ----------

 move.l #1006,d2                          ; Modus = New
 move.l #filename,d1                      ; Filename
 move.l dosbase,a6
 jsr -30(a6)                              ; OPEN ()
 move.l d0,filehd                         ; Fileadresse
 tst.l d0                                 ; Error ?
 bne ok7                                  ; wenn nicht, dann ok7
 move.l #text8,textanfang                 ; Errormeldung
 bsr meldung                              ; ausgeben
 bra exit                                 ; Programmende

ok7:
 move.l filehd,d1                         ; Fileadresse
 move.l #bob_header,d2                    ; Daten
 move.l #16,d3                            ; Länge = 16 Bytes
 move.l dosbase,a6
 jsr -48(a6)                              ; Write ()

 move.l filehd,d1                         ; ab hier
 move.l bob_planes_base,d2                ; werden die entpackten
 move.l bob_planes_size,d3                ; Grafikdaten vom BOB
 move.l dosbase,a6                        ; gespeichert
 jsr -48(a6)                              ; WRITE ()
 move.l filehd,d1
 move.l dosbase,a6
 jsr -36(a6)                              ; CLOSE ()

;---------- OK-Meldung ausgeben ----------

 move.l #text9,textanfang
 bsr meldung
 bra exit
```

```
;---------- Brush-Picdaten entpacken ----------

start_dekompression:
 move.l body_chunk,a0
 add.l #8.a0                    ; Quelle für Picdaten ist A0
 clr.l d7                       ; 7 = Zähler für eine Spalten-
                                ; anzahl
 clr.l d6                       ; Brushpicdatenzähler
 clr.l d5                       ; D5 = Zähler für Zeilenanzahl
 clr.l d4                       ; D4 = Zähler für Planes

deko_spalte:
 cmp.w bob_bytebreite,d7        ; schon eine Zeile einer Bitmap
                                ; entpacked ?
 bmi dekoa                      ; wenn nicht , dann `dekoa`
 clr.l d7                       ; SpaltenzäΣhler auf Null zurück-
                                  setzen
 add.w #1,d4                    ; Bitmapzähler um 1 erniedrigen
 cmp.w bob_planes.d4            ; schon eine komplette Zeile der
                                  Grafik
 bmi deko_spalte                ; entpacked ? - wenn nicht `de-
                                  ko_spalte`
 clr.l d4                       ; Planeszähler auf Null zurück-
                                  setzen
 cmp.l bob_planes_size,d6       ; schon komplette Grafik entpak-
                                  ket ?
 bmi deko_spalte                ; wenn nicht, dann weiter machen
 rts                            ; Ende der Entpackerroutine
```

```
;---------- Eine Zeile entpacken ----------

dekoa:
 move.l d4.d3                     ; Planenummer nach D3
 mulu #4,d3                       ; mit 4 multiplizieren
 move.l #bitmapadressen,a1        ; Ziel für Picdaten ist A1 (Ta-
                                    belle)
 move.l (a1,d3).a2                ; A2 ist Zeiger auf Bitmapadresse
 bsr bob_unpacker                 ; Eine Bitmapzeile entpacken
 move.l a2,(a1,d3)                ; Bitmapposition saven
 bra deko_spalte                  ; nächste Zeile

;---------- eigentliche Entpackerroutine ----------

bob_unpacker:
 clr.l d0                         ; Zähler  für max. 128 Bytes
 cmp.b #0,packerbyte              ; Brush gepacket ?
 bne gepacked                     ; wenn ja, dann `gepacked`

;---------- eine nicht gepackte Zeile übernehmen ----------

 move.w bob_bytebreite,d0         ; Bob-Breite in Bytes nach D0
 sub.w #1,d0                      ; minus 1
 bra deko1loop                    ; und Picdaten übernehmen

gepacked:
 move.b (a0)+,d0                  ; Befehlsbytes holen
 bmi deko2                        ; wenn negativ, dann `deko2`

deko1loop:
 move.b (a0)+,(a2)+               ; sonst Picdaten kopieren
 add.w #1,d7                      ; Spaltenzähler plus 1
 add.l #1,d6                      ; Brushdatenzähler plus 1
 dbra d0,deko1loop                ; solange bis Befehlsbyte = Null
 rts
```

```
deko2:
 neg.b d0                       ; vom Befehlsbyte Vorzeichen
                                  wechseln
 move.b (a0)+,d1                ; Füllbyte nach D1

deko2loop:
 move.b d1,(a2)+                ; Füllbyte kopieren
 add.w #1,d7                    ; solange bis Befehlsbyte
 add.l #1,d6                    ; gleich Null ist
 dbra d0,deko2loop
 rts

;---------- Programm beenden ----------

exit:
 move.l brush_base,a1           ; Wurde Speicher für
 cmp.l #0,a1                    ; Brush reserviert ?
 beq exit1                      ; wenn nicht, dann exit1
 move.l puffer+4,d0             ; sonst Speicher wieder
 move.l 4,a6                    ; freigeben
 jsr -210(a6)                   ; FreeMem ()

exit1:
 move.l bob_planes_base,a1      ; Wurde Speicher zum
 cmp.l #0,a1                    ; entpacken reserviert ?
 beq exit2                      ; wenn nicht, dann exit2
 move.l bob_planes_size,d0      ; sonst Speicher wieder
 move.l 4,a6                    ; freigeben
 jsr -210(a6)                   ; FreeMem ()

exit2:
 move.l dosbase,a1              ; dosbase
 move.l 4,a6
 jsr -414(a6)                   ; CloseLibrary ()
 rts
```

```
;---------- Text ausgeben ----------

meldung:
 move.l dosbase,a6                  ; aktives Window ermitteln
 jsr -60(a6)                        ; OutPut ()
 move.l d0,d1                       ; Windowadresse nach D1
 move.l textanfang,d2               ; Zeiger auf Text
 move.l d2,a0
 clr.l d3

loop:
 cmp.b #0,(a0)+
 beq loop1                          ; Textende ?
 add.l #1,d3                        ; Textzähler +1
 bra loop                           ; nächstes Zeichen

loop1:
 move.l dosbase,a6                  ; Textmeldung ausgeben
 jsr -48(a6)                        ; Write ()
 rts

;---------- Parameter ----------

textlänge: dc.l 0
textadresse: dc.l 0
textanfang: dc.l 0
dosname: dc.b "dos.library",0
 even
dosbase: dc.l 0
filehd: dc.l 0
puffer: blk.b 12,0
bob_header: blk.b 16,0
brush_base: dc.l 0
chunktabelle: dc.l "BMHD","BODY",0
chunkadresse: dc.l bmhd_chunk,body_chunk
iffchunk: dc.l 0
```

```
bmhd_chunk: dc.l 0
body_chunk: dc.l 0
planepick: dc.b 0
 even
planeonoff: dc.b 0
 even
bob_planes_size: dc.l 0
bob_planes_base: dc.l 0
bitmapadressen: blk.l 6,0
bob_planes: dc.w 0
bob_höhe: dc.w 0
bob_bytebreite: dc.w 0
packerbyte: dc.b 0                ; Dekompressions-Byte
 even
text1: dc.b "Bitte Disk worauf sich Brush befindet einlegen!",10,0
 even
text2: dc.b "Kann File auf Diskette nicht finden!",10,0
 even
text3: dc.b "Kein IFF-Standart!",10,0
 even
text4: dc.b "Datei nicht im ILBM-Format!",10,0
 even
text5: dc.b "Nicht genug Speicher vorhanden!",10,0
 even
text6: dc.b "Brush erfolgreich in BOB umgewandelt!",10,0
 even
text7: dc.b "Bitte  Disk  einlegen,  worauf  BOB  gespeichert  werden
            kann",10,0
 even
text8: dc.b "Disk-Error!",10,0
 even
text9: dc.b "BOB erfolgreich auf Disk gespeichert!",10,0
 even
filename: dc.b "TransBOB",0
 even
 END
```

Anhang A

Strukturen im Überblick

Anhang A

Strukturen im Überblick

In diesem Anhang finden sie alle Strukturen im Uebcrblick, welche im Buch verwendet wurden (alphbetisch geordnet):

'AnimOb'-Struktur: (Länge = 40 Bytes)

Die AnimOb-Struktur enthält ein Animationsobjekt, welches aus verschiedenen AnimComps besteht, in seiner Gesamtheit. Dieses Objekt wird mit 'AddAnimOb ()' dem System zugänglich gemacht und kann mit 'Animate ()' animiert werden. 'SortGList ()' und 'DrawGList ()' printen dann das Objekt auf den Bildschirm.

Offset:	Typ:	Bezeichnung:	Erklärung:
;---- es folgen als erstes Systemvariablen -----			
000	Long	NextOb	Zeiger auf nächste 'AnimOb'-Struktur (oder Null)
004	Long	PrevOb	Zeiger auf Vorgängerstruktur (s.o.)
008	Long	Clock	enthält Anzahl 'Animate ()'- Aufrufe die dieses Objekt schon erlebt hat
012	Word	AnOldY	alte y-Position
014	Word	AnOldX	alte x-Position
;--- nun folgen die User-Variablen -----			
016	Word	AnY	y-Position (Achtung - siehe unten)

018	Word	AnX	x-Position (Achtung - siehe unten)
020	Word	YVel	Geschwindigkeit in y-Richtung (siehe unten)
022	Word	XVel	Geschwindigkeit in x-Richtung (siehe unten)
024	Word	YAccel	y-Beschleunigung (siehe unten)
026	Word	XAccel	x-Beschleunigung (siehe unten)
028	Word	RingYTrans	steigende Beschleunigung in Y-Richtung (s. u.)
030	Word	RingXTrans	steigende Beschleunigung in X-Richtung (s. u.)
032	Long	AnimORoutine	siehe unten
036	Long	HeadComp	Zeiger auf erste 'AnimComp'-Struktur
040		END	Ende der 'AnimOb'-Struktur

AnimComp-Struktur: (Länge = 38 Bytes)

Die AnimComp-Struktur (Animationskomponente) stelt die Verbindung zwischen BOB und Animationsobjekt (AnimOb-Struktur) her. Besonders bei der Sequenzanimation (RingTrigger) ist sie von großer Wichtigkeit, denn sie verbindet einzelne Komponenten zu einem Ring miteinander und bestimmt, wie lange jede einzelne Sequenz aktiv sein soll.

Offset:	Typ:	Bezeichnung:	Erklärung:
000	Word	CompFlags	Bit 0 = 1 , dann wird mit hilfe der Zeiger 'NextSeq' und 'PrevSeq' eine Ringsequenz vom System durchgeführt. (RingTrigger-Flag)
002	Word	Timer	wird mit Wert aus TimeSet geladen und auf Null heruntergezählt und dann falls gewünscht (Ring-Trigger-Flag = 1) die nächste Sequenz dargestellt
004	Word	TimeSet	wie lange eine Sequenz dargestellt werden soll
006	Long	NextComp	Zeiger auf nächste AnimComp-Struktur (s. u.)
010	Long	PrevComp	Zeiger auf Vorgänger-AnimComp-Struktur (s.u.)
014	Long	NextSeq	Zeiger auf nächste AnimComp-Struktur (s. u.)
018	Long	PrevSeq	Zeiger auf Vorgänger-AnimComp-Struktur (s.u.)
022	Long	AnimCRoutine	siehe 'AnimORoutine'
026	Word	Y-Trans	y-Positon der AnimComp-Struktur, in 64stel
028	Word	X-Trans	X-Postion der AnimComp-Struktur, in 64stel
030	Long	HeadOb	Zeiger auf dazugehörige 'AnimOb'-Struktur
034	Long	AnimBob	Zeiger auf dazugehörige BOB-Struktur

038		END	Ende der AnimComp-Struktur

'Bitmap'-Struktur: (Länge = 40 Bytes)

Offset:	Typ:	Bezeichnung:	Beschreibung:
000	Word	BytePerRow	Bytes pro Display-Zeile
002	Word	Rows	Anzahl der Display-Zeilen
004	Byte	Flags	System-Flags
005	Byte	Depth	Anzahl der Bitmaps (Tiefe)
006	Word	Pad	unbenutzt
008	Long	PlanePtr1	Zeiger auf 1. Bitmap (02 Farben)
012	Long	PlanePtr2	Zeiger auf 2. Bitmap (04 Farben)
016	Long	PlanePtr3	Zeiger auf 3. Bitmap (08 Farben)
020	Long	PlanePtr4	Zeiger auf 4. Bitmap (16 Farben)
024	Long	PlanePtr5	Zeiger auf 5. Bitmap (32 Farben)
028	Long	PlanePtr6	Zeiger auf 6. Bitmap (HAM)
032	Long	PlanePtr7	Zeiger auf 7. Bitmap (unbenutzt)
036	Long	PlanePtr8	Zeiger auf 8. Bitmap (unbenutzt)
040		END	Ende der 'Bitmap'-Struktur

'BOB'-Struktur: (Länge = 30 Bytes)

Muß mit 'VSprite'-Struktur verbunden werden (Offset 52)!

Offset:	Typ:	Bezeichnung:	Beschreibung:
000	Word	B_Flags	wie das BOB vom System behandelt werden soll:
	User--SaveBOB		Bit 0 = 1, dann wirkt BOB wie ein Paintbrush
	User--BOBisComp		Bit 1 = 1, Bob besitzt eine 'AnimComp'-Struktur
	BWaiting		Bit 8 = 1, ein BOB auf den 'Before' in der 'BOB'-Struktur zeigt, wird zuerst gezeichnet
	BDrawn		Bit 9 = 1, BOB wurde gezeichnet
	BobsAway		Bit10 = 1. BOB wird beim nächsten 'DrawGList ()' aus dem Gelsystem entfernt
	BobNix		Bit11 = 1, Bob wurde aus dem Gelsystem entfernt
002	Long	SaveBuffer	Zeiger auf Puffer zum speichern des Hintergrundes (nur wenn SaveBack-Flag in VSprite gesetzt ist) SaveBufferSize = bobhöhe * bobbreite * 2 * screentiefe
006	Long	ImageShadow	Zeiger auf Puffer in der die Maske aller gesetzten Bits von allen Bitmaps zusammen steht ImageShadowSize = bobhöhe * bobbreite * 2

010	Long	Before	Zeiger auf BOB-Struktur, die über diesen BOB gezeichnet werden soll (oder Null) - Priorität
014	Long	After	Zeiger auf BOB-Struktur, die hinter diesem BOB gezeichnet werden soll (oder Null) - Priorität
018	Long	VSprite	Zeiger auf dazugehörige VSprite-Struktur
022	Long	BobComp	Zeiger auf 'AnimComp'-Struktur, nur wenn Flag 'BOBisComp' in BOB-Struktur gesetzt (sonst Null)
026	Long	DBuffer	Zeiger auf 'DBufPakket'-Struktur (wenn BOB doppelt gepuffert werden soll - sonst Null)
030		END	Ende der 'BOB'-Struktur

'ColorMap'-Struktur: (Länge = 8 Bytes)

Offset:	**Typ:**	**Bezeichnung:**	**Beschreibung:**
000	Byte	Flag	System-Flag
001	Byte	Typ	System
002	Word	Count	Anzahl Farben (max. 32)
004	Long	ColorTable	Zeiger auf einen Puffer mit 32 Words (64 Bytes) in dem die Farbwerte stehen
008		END	Ende der 'ColorMap'-Struktur

'DBufPacket'-Struktur: (Länge = 16 Bytes)

Offset:	Typ:	Bezeichnung:	Beschreibung:
000	Word	BufY	Y-Position vom Hintergrund des zweiten Screens
002	Word	BufX	X-Position vom Hintergrund des zweiten Screens
004	Long	BufPath	Zeiger auf VSprite-Struktur
008	Long	BufBuffer	Zeiger auf einen Puffer, der genausogroß ist, wie der von SaveBuffer, zum Speichern des zweiten Hintergrundes
012	Long	BufPlanes	Zeiger auf Planepointers des zweiten Screens
016		END	Ende der DBufPacket-Struktur

'GelsInfo'-Struktur: (Länge = 38 Bytes)

'InitGels ()' füllt diese Struktur mit den wichtigsten Werten!

Offset:	Typ:	Bezeichnung:	Beschreibung:
000	Byte	SprRsrvd	welche Sprites als VSprites genutzt werden
001	Byte	Flags	System-Flag
002	Long	GelHead	Zeiger auf erste 'VSprite'-Struktur

006	Long	GelTail	Zeiger auf letzte 'VSprite'-Struktur
010	Long	NextLine	Zeiger auf Puffer mit 8 Words (nur bei VSprites)
014	Long	LastColor	Zeiger auf Puffer mit 8 Longs (nur bei VSprites)
018	Long	CollHandler	Zeiger auf Puffer mit 16 Longs (Adressen der 16 verschiedenen Collisionsroutinen)
022	Word	Leftmost	linker Begrenzungsrand (für Collisions-Routine 0)
024	Word	Rightmost	rechter Begrenzungsrand (für Collisions-Routine 0)
026	Word	Topmost	oberer Begrenzungsrand (für Collisions-Routine 0)
028	Word	Bottommost	unterer Begrenzungsrand (für Collisions-Routine 0)
030	Long	FirstBlissObj	wird nur vom System benutzt
034	Long	LastBlissObj	wird nur vom System benutzt
038		END	Ende der 'GelsInfo'-Struktur

'Interrupt'-Struktur: (Länge = 22 Bytes)

Offset:	**Typ:**	**Bezeichnung:**	**Beschreibung:**
000	Long	Succ	Zeiger auf nächste Liste (unwichtig)
004	Long	Pred	Zeiger auf vorangegangene Liste (unwichtig)

008	Byte	Type	kodierter Node-Typ (auf Wert 2 setzen)
009	Byte	Pri	Priorität dieser Liste (von +127 bis -128)
010	Long	Name	Zeiger auf Namen dieser Liste (unwichtig)
014	Long	Data	Zeiger auf Datenspeicher (unwichtig)
018	Long	Code	Zeiger auf Interruptrouti-ne
022		END	Ende der Interrupt-Struk-tur

'IText'-Struktur: (Länge = 20 Bytes)

Offset:	Typ:	Bezeichnung:	Beschreibung:
000	Byte	DetailPen	TextFarbe (Farbregister-nummer)
001	Byte	BlockPen	Hintergrundfarbe (Farbre-gisternummer)
002	Byte	Mode	Darstellungsart (0, 1, 4, 5, 8)
003	Byte	Pad	unbenutzt
004	Word	x_pos	relative X-Position zur aktuellen Position
006	Word	y_pos	relative Y-Position zur aktuellen Position
008	Long	Font	Zeiger auf TextFont-Struktur oder Null
012	Long	Text	Zeiger auf Text, welcher mit Null endet
016	Long	NextIText	Zeiger auf weitere IText-Struktur oder Null

020		END	Ende der IText-Struktur

'OSArgs'-Struktur: (Länge = 32 Bytes)

Wird für die Routine 'OpenScreen ()' benötigt!

Offset:	Typ:	Bezeichnung:	Beschreibung:
000	Word	X_Pos	X-Position des Screens im ViewPort (Normal = 0)
002	Word	Y_Pos	Y-Position des Screens im ViewPort
004	Word	Width	Breite der Bitmap
006	Word	Heigth	Höhe der Bitmap
008	Word	depth	Anzahl der Bitplanes (tiefe)
010	Byte	detail_pen	Farbregisternummer für Textfarbe
011	Byte	block_pen	Farbregisternummer für Hintergrundfarbe
012	Word	View_Modes	siehe ViewPort-Struktur
014	Word	Screen_Typ	siehe unten
016	Long	Font	Zeiger auf neuen Zeichensatz (0 = Normal)
020	Long	title	Zeiger auf Titletext, der mit Null endet
024	Long	Gadget	Zeiger auf Gadget-Struktur (0 = keine)
028	Long	BitMap	Zeiger auf eigene Bitmap-Struktur (nur wenn in Screen_Typ = Bit7 gesetzt ist, sonst Wert Null)
032		END	Ende der OSArgs-Struktur

Screen_Typ-Flag:

Bit:	Wert:	Bezeichnung:	Beschreibung:
0	$001	WBenchScreen	dies ist der Workbench-Screen
0-3	$00f	CustomScreen	Screen mit allen Funktionen
4	$010	ShowTitle	Bit wird gesetzt, wenn man 'ShowTitle ()' aufruft
5	$020	Beeping	Bit wird gesetzt, wenn man 'DisplayBeep ()' aufruft
6	$040	CustomBitMap	muß man setzen, wenn an seine eigene 'BitMap'-Struktur benutzen will
7	$080	ScreenBehind	wenn gesetzt, Screen wird hinter allen anderen Screens geöffnet
8	$100	ScreenQuiet	es erscheinen keine Gadgets und kein Title mehr auf dem Screen

---

'OWArgs'-Struktur: (Länge = 48 Bytes)

Wird für die Routine 'OpenWindow ()' benötigt!

Offset:	Typ:	Bezeichnung:	Beschreibung:
000	Word	X_Pos	X-Position des Windows auf dem Screen
002	Word	Y_Pos	Y-Position des Windows auf dem Screen

004	Word	Width	Breite des Windows
006	Word	Heigth	Höhe des Windows
008	Byte	detail_pen	Farbregisternummer für Textfarbe
009	Byte	block_pen	Farbregisternummer für Hintergrundfarbe
010	Long	IDCMP-Flags	siehe unten
014	Long	Window_Typ	siehe unten
018	Long	Gadget	Zeiger auf Gadget-Struktur (0 = keine)
022	Long	CheckMark	Zeiger auf eine Graphikstruktur für eigene Symbole zum Abhaken von Menüpunkten
026	Long	title	Zeiger auf Windowtitletext, der mit Null endet
030	Long	Screen	Zeiger auf eine 'Screen'-Struktur
034	Long	BitMap	Zeiger auf eigene Bitmap-Struktur (0 = keine)
038	Word	MinWidth	Mindestbreite des Fensters
040	Word	MinHeigth	Mindesthöhe des Fensters
042	Word	MaxWidth	Maximalbreite des Fensters
044	Word	MaxHeigth	Miximalhöhe des Fensters
046	Word	Screen_Typ	muß der selbe wie beim Screen sein
048		END	Ende der OWArgs-Struktur

IDCMP-Flags:

diese Bits bestimen, bei welchen Ereignissen Intuition dem Programm eine Meldung übermitteln soll!

Bit:	Wert:	Bezeichnung:	Nachricht bei:
00	$000001	SizeVerify	
01	$000002	NewSize	Veränderung der Fenstergröße
02	$000004	RefreshWindow	erneuern eines Windows
03	$000008	MouseButtons	drücken einer Maustaste
04	$000010	MouseMove	Bewegungen der Maus
05	$000020	GadGetDown	Auswahl eines speziellen Gadgets
06	$000040	GadgetUp	s.o. (aber wenn linke Maustaste losgelassen wird)
07	$000080	ReqSet	Erscheinung eines Requesters
08	$000100	MenuPick	Auswahl eines Menüpunktes
09	$000200	CloseWindow	schließen des Fensters
10	$000400	RawKey	drücken einer Taste
11	$000800	ReqVerify	
12	$001000	ReqClear	verschwinden eines Requesters
13	$002000	MenuVerify	
14	$004000	NewPrefs	ändern der Preferences
15	$008000	DiskInserted	einlegen einer Diskette
16	$010000	DiskRemoved	Herausnehmen einer Diskette
17	$020000	WBenchMessage	
18	$040000	ActiveWindow	aktivieren des Windows
19	$080000	InActiveWindow	desaktivieren des Windows

20	$100000	DeltaMove	Mausbewegungen relativ melden
21	$200000	VanillaKey	
22	$400000	IntuiTicks	

Window_Typ:

die Bits bestimmen die Eigenschaften eines Fensters!

Bit:	**Wert:**	**Bezeichnung:**	**Beschreibung:**
00	$0000001	WindowSizing	Fenstergröße veränderbar
01	$0000002	WindowDrag	Fenster verschiebbar
02	$0000004	WindowDepth	Fensterüberlagerung möglich
03	$0000008	WindowClose	besitzt ein Fensterschließsymbol
04	$0000010	SizeBright	Vergrößerungsgadget ist rechts
05	$0000020	SizeBBottom	Vergrößerungsgadget ist unten
06	$0000040	SimpleRefresh	Neuzeichnen manuell
07	$0000080	SuperBitMap	ganzen Fensterinhalt speichern
08	$0000100	BackDrop	Fenster nach Hinten
09	$0000200	ReportMouse	Mauskoordinaten melden
10	$0000400	GimmeZeroZero	Fenster ohne Leiste
11	$0000800	BorderLess	Fenster ohne Ränder
12	$0001000	Activate	Fenster sofort aktiv
13	$0002000	WindowAktive	wird von Intuition gesetzt
14	$0004000	InRequest	wird von Intuition gesetzt
15	$0008000	MenuState	wird von Intuition gesetzt
16	$0010000	RMBTrap	bei drücken der rechten Maustaste kein Menü

17	$0020000	NoCareRefresh	keine Erneuerungsmeldung
24	$1000000	WindowRefresh	wird von Intuition gesetzt
25	$2000000	WBenchWindow	wird von Intuition gesetzt
26	$4000000	WindowTicked	wird von Intuition gesetzt

'RasInfo'-Struktur: (Länge = 12 Bytes)

Offset:	Typ:	Bezeichnung:	Beschreibung:
000	Long	Next	Zeiger auf 2. 'RasInfo'-Struktur (nur wenn DualPlayfield benutzt wird)
004	Long	Bitmap	Zeiger auf 'Bitmap'-Struktur
008	Word	RxOffset	X-Position in der Bitmap (Normal =0) - für größere Bitmaps, um z.B. Scrolling zu erzeugen
010	Word	RyOffset	Y-Position in der Bitmap (Normal =0) s.o.
012		END	Ende der 'RasInfo'-Struktur

'RastPort'-Struktur (Länge = 100 Bytes)

Offset:	Typ:	Bezeichnung:	Beschreibung:
000	Long	Layer	Zeiger auf 'Layer'- Struktur
004	Long	BitMap	Zeiger auf 'BitMap'-Struktur
008	Long	AreaPtrn	Zeiger auf das AreaFill-Muster
012	Long	TmpRas	Zeiger auf 'TmpRas'-Struktur
016	Long	AreaInfo	Zeiger auf 'AreaInfo'-Struktur
020	Long	GelsInfo	Zeiger auf 'GelsInfo'-Struktur
024	Byte	Mask	Schreibmaske (wieviel Bitmaps zugelassen sind)
025	Byte	FgPen	Farbregisternummer für Vordergrundfarbe
026	Byte	BgPen	Farbregisternummer der Hintergrundfarbe
027	Byte	AOlPen	Farbregisternummer der AreaFill-Outlinefarbe
028	Byte	DrawMode	ZeichenModi (0=Jam1, 1=Jam2, 2=Complement, 3=Invers) für Routine 'SetDrMd ()'
029	Byte	AreaPtSz	Höhe des AreaFill-Musters (2^n Words))
030	Byte	LinePatCNT	unbenutzt
031	Byte	Dummy	Line Draw Pattern Preshift
032	Word	Flags	verschiedene Kontrollbits (FirstDot,OneDot etc.)
034	Word	LinePtrn	16 Bits für Linienmuster

036	Word	CP_X	x-Position des Grafikcursors
038	Word	CP_Y	y-Position des Grafikcursors
040	Byte	Minterms(8)	8 x 1 Byte für Minterms (Funktion unbekannt)
048	Word	PenWidth	Cursorbreite
050	Word	PenHeight	Cursorhöhe
052	Long	TextFont	Zeiger auf 'TextFont'-Struktur
056	Byte	AlgoStyle	Zeichensatzmodus (Style-Flag)
057	Byte	TxFlags	textspezifische Flags
058	Word	TxHeight	Höhe des Zeichensatzes
060	Word	TxWidth	durchschnittliche Zeichenbreite
062	Word	TxBaseLine	Texthöhe ohne Unterlängen (für 'Text'-Routine - Wert zu Y-Position addieren = genaue Y-Pos.)
064	Word	TxSpacing	Zeichenabstand
066	Long	RP-User	Zeiger auf evtuelle Userdaten (unwichtig!)
070	Word		7 Words reserviert
084	Long		2 Longs reserviert
092	Byte		8 Bytes reserviert
100		END	Ende der Datenstruktur

'Screen'-Struktur: (Länge = 342 Bytes)

Erhält man in D0 zurück, wenn man 'OpenScreen ()' aufruft!

Offset:	Typ:	Bezeichnung:	Beschreibung:
000	Long	NextScreen	Zeiger auf nächsten Screen
004	Long	FirstWindow	Zeiger auf erstes Window auf diesem Screen
008	Word	X_Pos	x-Position des Screens
010	Word	Y_Pos	y-Position des Screens
012	Word	Width	Breite der Bitmap
014	Word	Heigth	Höhe der Bitmap
016	Word	MouseY	Y-Koordinate des Mauspointers
018	Word	MouseX	X-Koordinate des Mauspointers
020	Word	ScreenTyp	siehe Screen_Typ
022	Long	Title	Zeiger auf Screen-Titletext
026	Long	STitle	Zeiger auf Standart Titletext
030	Byte		Kopfleistenhöhe
031	Byte		vertikale Grenze der Kopfleiste
032	Byte		horizontale Grenze der Kopfleiste
033	Byte		vertikale Grenze des Menüs
034	Byte		horizontale Grenze des Menüs
035	Byte		oberer Fensterrahmen
036	Byte		linker Fensterrahmen
037	Byte		rechter Fensterrahmen
038	Byte		unterer Fensterrahmen

039	Byte	Pad	unbenutzt
040	Long	Font	Zeiger auf Font-Struktur
040	----	ViewPort	ab hier liegt die View-Port-Struktur vom Screen
084	----	RastPort	ab hier liegt die Rast-Port-Struktur vom Screen
184	----	BitMap	ab hier liegt die BitMap-Struktur vom Screen
224	----	LayerInfo	ab hier liegt die LayerIn-fo-Struktur vom Screen
326	Long	GadGet	Zeiger auf erste Screen-Gadget-Struktur
330	Byte	DetailPen	Farbregisternummer für die Schriftfarbe
331	Byte	BlockPen	Farbregisternummer für die Hintergrundfarbe
332	Word	BackUp	Backup-Register für 'DisplayBeep ()'-Routine
334	Long	Extern	Zeiger auf externe Daten
338	Long	User	Zeiger auf User Daten
342		END	Ende der 'Screen'- Struktur

'SimpleSprite'-Struktur: (Länge = 12 Bytes)

Offset:	**Typ:**	**Bezeichnung:**	**Beschreibung:**
000	Long	Datas	Zeiger auf Spritedaten
004	Word	Höhe	Höhe des Sprites
006	Word	X	X-Position des Sprites
008	Word	Y	Y-Position des Sprites
010	Word	Nummer	Nummer des Sprites (0-7)
012		END	Ende der SimpleSprite-Struktur

'TextAttr'-Struktur: (Länge = 8 Bytes)

Offset:	Typ:	Bezeichnung:	Beschreibung:
000	Long	FontName	Zeiger auf Fontnamen, welcher mit Null endet
004	Word	Height	Fonthöhe (steht im jeweiligen Fonts-Dir)
006	Byte	Style	Stil des Fonts (siehe TextFont-Struktur)
007	Byte	Flags	Preferences (siehe TextFont-Struktur)
008		END	Ende der TextAttr-Struktur

'TextFont'-Struktur: (Länge = 52 Bytes)

Offset:	Typ:	Bezeichnung:	Beschreibung:
000	Long	Succ	Zeiger auf nächsten Font
004	Long	Pred	Zeiger auf vorrangegangenen Font
008	Byte	Type	Node-Typ (12 = Font)
009	Byte	Pri	Priorität
010	Long	Name	Zeiger auf Fontnamen
014	Long	ReplyPort	Zeiger auf AntwortPort
018	Word	Lenght	Länge der Message
020	Word	YSize	Höhe des Zeichensatzes
022	Byte	Style	Stil des Zeichensatzes:

Normal	normal	= 0
Underline	unterstrichen	= 1
Bold	fett	= 2
Italic	kursiv	= 4
	extended	= 8

023	Byte	Flags	Preferences und Flags: ROM-Font = 1 Disk-Font = 2 Rev-Path = 4 TallDot = 8 WideDot = 16 Proportional = 32 Designed = 64 Removed = 128
024	Word	XSize	durchschnittliche Fontbreite
026	Word	Baseline	Fonthöhe ohne Unterlängen
028	Word	BoldSmear	Smear-Effekt für Fettdruck
030	Word	Accessors	Zugriffszähler (Null = Font wird gelöscht)
032	Byte	LoChar	ASCII-Code des ersten Zeichens
033	Byte	HiChar	ASCII-Code des letzten Zeichens
034	Long	CharData	Zeiger auf Zeichensatzdaten
038	Word	Modulo	Bytes pro Zeichensatz-Zeile
040	Long	CharLoc	Zeiger auf Offset-Daten für Zeichendecodierung
044	Long	CharSpace	Zeiger auf Breiten-Tabelle der Zeichen
048	Long	CharKern	Zeiger auf Zeichenkern der Tabelle
052		END	Ende der TexFont-Struktur

'UCopList'-Struktur (Länge = 12 Bytes - Mit 'AllocMem ()' reservieren):

Offset:	Typ:	Bezeichnung:	Beschreibung:
000	Long	NextUCopList	Zeiger auf nächste User-Copperlist-Struktur
004	Long		Zeiger auf erste Copperliste die von 'MakeVPort ()' angelegt wurde
008	Long		Zeiger auf dieser Struktur zugehörigen Copperliste
012		END	Ende der 'UCopList'-Struktur

'View'-Struktur: (Länge = 18 Bytes)

Offset:	Typ:	Bezeichnung:	Beschreibung:
000	Long	ViewPort	Zeiger auf erste 'View-Port'-Struktur
004	Long	LOFCprList	Zeiger auf LongFrame-Copperliste wird mit 'MrgCop ()' erzeugt
008	LONG	SHFCprList	Zeiger auf ShortFrame-Copperliste
012	Word	DyOffset	y-Position des Displays (wird vom System init.)
014	Word	DxOffset	x-Position des Displays (wird vom System init.)
016	Word	Modes	Modus-Flag (Inerlace-Flag hier setzen, falls gewünscht)
018		END	Ende der 'View'-Struktur

'ViewPort'-Struktur: (Länge = 40 Bytes)

Offset:	Typ:	Bezeichnung:	Beschreibung:
000	Long	Next	Zeiger auf nächste 'ViewPort'-Struktur
004	Long	ColorMap	Zeiger auf 'ColorMap'-Struktur
008	Long	DspIns	Zeiger auf Display-Copperliststruktur
012	Long	SprIns	Zeiger auf Sprite-Copperliststruktur1
016	Long	ClrIns	Zeiger auf Sprite-Copperliststruktur2
020	Long	UCopList	Zeiger auf 'UCopList'-Struktur (fr User)
024	Word	DWidth	Breite des Displays (Screen) in Pixel
026	Word	DHeight	Höhe des Displays (Screen) in Zeilen
028	Word	DxOffset	X-Pos. des Displays (Screen)
030	Word	DyOffset	Y-Pos. des Displays (Screen)
032	Word	Modes	ViewPort-Modi:

```
               Wert $0000 = Normal
    Bit  1=1   Wert $0002 = GenlockVideo
    Bit  2=1   Wert $0004 = Interlace
    Bit  6=1   Wert $0040 = PFBA
    Bit  7=1   Wert $0080 = Extra-Halfbrite
    Bit  8=1   Wert $0100 = GenlockAudio
    Bit 10=1   Wert $0400 = DualPlayField
    Bit 11=1   Wert $0800 = Hold and Modify
    Bit 13=1   Wert $2000 = VP-Hide
    Bit 14=1   Wert $4000 = Sprites
    Bit 15=1   Wert $8000 = Hires (Breite 640)
```

034	Word	reserved	reserviert
036	Long	RasInfo	Zeiger auf 'RasInfo'-Struktur
040		END	Ende der 'ViewPort'-Struktur

'VSprite'-Struktur für Sprites: (Länge = 58 Bytes)

Offset:	**Typ:**	**Bezeichnung:**	**Beschreibung:**
000	Long	NextVSprite	Zeiger auf nächste 'VSprite'-Struktur
004	Long	PrevVSprite	Zeiger auf Vorgänger 'VSprite'-Struktur
008	Long	DrawPath	wird nur vom System benutzt
012	Long	ClearPath	wird nur vom System benutzt
016	Word	OldY	alte y-Position vom VSprite (System-Variable)
018	Word	OldX	alte x-Position vom VSprite (System-Variable)
020	Word	Flags	beschreibt wie das GEL zu behandeln ist:
	VSprite	Bit 0 = 1,	dann Gel ist ein VSprite sonst BOB
	SaveBack	Bit 1 = 1,	Hintergrund wird gespeichert
	Overlay	Bit 2 = 1,	nur gesetzte Bits des Gels werden in die Grafik kopiert

	MustDraw	Bit 3 = 1,	für VSprites (Farben speichern)
	BackSaved	Bit 8 = 1,	Hintergrund wurde Gespeichert (System)
022	Word	Y	Y-Position des VSprites
024	Word	X	X-Position des VSprites
026	Word	Heigth	Höhe des VSprites (Anzahl Zeilen)
028	Word	Width	Breite des VSprites (Anzahl in Words)
030	Word	Depth	Anzahl Bitplanes (Tiefe)
032	Word	MeMask	siehe Kapitel Collisionen
034	Word	HitMask	siehe Kapitel Collisionen
036	Long	ImageData	Zeiger auf Grafikdaten vom VSprite (Datas)
040	Long	Borderline	siehe Kapitel Collisionen
044	Long	CollMask	siehe Kapitel Collisionen
048	Long	SprColors	Zeiger auf Puffer mit 4 Words für Farbspeicherung
052	Long	Bob	Zeiger auf 'BOB'-Struktur, (bei VSprites = 0)
056	Byte	PlanePick	unwichtig (auf Null setzen)
057	Byte	PlaneOnOff	unwichtig (auf Null setzen)
058		END	Ende der 'VSprite'-Struktur

'VSprite'-Struktur für BOBs: (Länge = 58 Bytes)

Offset:	Typ:	Bezeichnung:	Beschreibung:
000	Long	NextVSprite	Zeiger auf nächste 'VSprite'-Struktur
004	Long	PrevVSprite	Zeiger auf Vorgänger 'VSprite'-Struktur
008	Long	DrawPath	wird nur vom System benutzt
012	Long	ClearPath	wird nur vom System benutzt
016	Word	OldY	alte y-Position vom BOB (System-Variable)
018	Word	OldX	alte x-Position vom BOB (System-Variable)
020	Word	Flags	beschreibt wie das GEL zu behandeln ist:
	SaveBack	Bit 1 = 1.	Hintergrund wird gespeichert
	Overlay	Bit 2 = 1.	nur gesetzte Bits des BOBs werden in die Grafik kopiert
022	Word	Y	Y-Position des BOBs
024	Word	X	X-Position des BOBs
026	Word	Heigth	Höhe des BOBs (Anzahl Zeilen)
028	Word	Width	Breite des BOBs (Anzahl in Words)
030	Word	Depth	Anzahl Bitplanes (Tiefe)
032	Word	MeMask	siehe Kapitel Collisionen
034	Word	HitMask	siehe Kapitel Collisionen
036	Long	ImageData	Zeiger auf Grafikdaten vom BOB (Datas)
040	Long	Borderline	siehe Kapitel Collisionen

			(Wichtig)
044	Long	CollMask	siche Kapitel Collisionen
048	Long	SprColors	unwichtig
052	Long	Bob	Zeiger auf 'BOB'-Struktur
056	Byte	PlanePick	siehe unten
057	Byte	PlaneOnOff	siehe unten
058		END	Ende der 'VSprite'-Struktur

'Window'-Struktur: (Länge = 124 Bytes)

Erhält man in D0 zurück, wenn man 'OpenWindow ()' aufruft!

Offset:	Typ:	Bezeichnung:	Beschreibung:
000	Long	NextWindwo	Zeiger auf nächste Fensterstruktur
004	Word	x-Pos	x-Position des Windows, relativ zum Screen
006	Word	y-Pos	y-Position des Windows, relativ zum Screen
008	Word	Width	Breite des Fensters
010	Word	Heigth	Höhe des Fensters
012	Word	DeltaMouseY	Y-Koordinate der Maus, relativ zum Fenster
014	Word	DeltaMouseX	X-Koordinate der Maus, relativ zum Fenster
016	Word	MinWidth	minimale Breite des Fensters
018	Word	MinHeigth	minimale Höhe des Fensters
020	Word	MaxWidth	maximale Breite des Fensters

022	Word	MaxHeigth	maximale Höhe des Fensters
024	Long	WindowTyp	siehe oben (Window_Typ)
028	Long	Menu	Zeiger auf Menüstruktur
032	Long	Title	Zeiger auf TitleText
036	Long	FirstRequest	Zeiger auf erste aktive Requesterstruktur
040	Long	DKRequest	Zeiger auf Double-Click-Requesterstruktur
044	Word	BRequest	Anzahl der Requester, die das Fenster sperren
046	Long	Screen	Zeiger auf 'Screen'-Struktur
050	Long	RastPort	Zeiger auf 'RastPort'-Struktur des Fensters
054	Byte		Linker Rahmen
055	Byte		Oberer Rahmen
056	Byte		Rechter Rahmen
057	Byte		Unterer Rahmen
058	Long	RRastPort	Zeiger auf Rahmen-RastPort-Struktur
062	Long	Gadget	Zeiger auf erste Gadget-Struktur
066	Long		Zeiger auf Eltern-Fenster-Struktur
070	Long		Zeiger auf Kind-Fenster-Struktur
074	Long	SprData	Zeiger auf SpriteData für MausPointer
078	Byte	SprHeigth	Höhe des Spritepointers
079	Byte	SprWidth	Breite des Spritepointers (max. 16)
080	Byte	XOffset	x-Offset des Spritepointers
081	Byte	YOffset	y-Offset des Spritepoin-

			ters
082	Long	IDCMP-Flag	siehe oben
086	Long	UMessagePort	User-Message-Port
090	Long	WMessagePort	Fenster-Message-Port
094	Long	MessageKey	Intuition-Message-Port
098	Byte	DetailPen	Farbregisternummer der Schriftfarbe
099	Byte	BlockPen	Farbregisternummer der Hintergrundfarbe
100	Long		Zeiger auf eigene Menü-Hakensymbole-Struktur
104	Long	STitle	Zeiger auf Screen-Title-text
108	Word	GZZ-MausX	
110	Word	GZZ-MausY	
112	Word	GZZ-Width	
114	Word	GZZ-Heigth	
116	Long	Extern	Zeiger auf Externe Daten
120	Long	User	Zeiger auf User Daten
124		END	Ende der 'Window'-Struktur

Anhang B

Library-Funktionen im Überblick

Anhang B - Library-Funktionen im Überblick

In diesem Anhang finden sie alle Routinen im Überblick, welche im Buch verwendet wurden (alphabetisch geordnet):

```
Routine   :  AddAnimOb (AnimOb,HeadOb,RastPort) (A0,A1,A2)
Library   :  graphics.library
Offset    :  -156 = -$9c
Parameter:   A0 =  Zeiger auf angelegte `AnimOb`-Struktur
             A1 =  Zeiger auf einen Puffer von 4 Bytes (dc.l 0). Die-
                   ser Puffer zeigt immer auf das zuletzt eingefügte
                   `AnimOb`. Beim ersten Aufruf dieser Routine, muß
                   dieser Puffer mit Null gefüllt sein.
             A2 =  Zeiger auf `RastPort`-Struktur
Rückgabe :   keine
Erklärung:   Diese Routine fügt ein Objekt zum Gelsystem hinzu.
```

```
Routine   :  AddBob (BOB,RastPort) (A0,A1)
Library   :  graphics.library
Offset    :  -96 = -$60
Parameter:   A0 =  Zeiger auf angelegte `BOB`-Struktur
             A1 =  Zeiger auf `RastPort`-Struktur
Rückgabe :   keine
Erklärung:   Diese Routine führt einen BOB zum Gelsystem hinzu. `BOB`-
             und `VSprite`-Struktur müssen vorher miteinander verbun-
             den werden!
```

```
Routine   :  AddIntServer (Nummer,Interrupt) (D0,A1)
Library   :  exec.library
Offset    :  -168 = -$A8
Parameter:  D0 =  Interruptnummer - welcher IRQ benutzt werden soll
                  (Erlaubt sind: 3, 5, 13, 15)
            A1 =  Zeiger auf Interrupt-Struktur (siehe oben)
Rückgabe :  keine
Erklärung:  Diese Routine fügt einen Interrupt zum System hinzu
```

```
Routine   :  AddVSprite (VSprite,RastPort) (A0,A1)
Library   :  graphics.library
Offset    :  -102 = -$66
Parameter:  A0 =  Zeiger auf angelegte `VSprite`-Struktur
            A1 =  Zeiger auf `RastPort`-Struktur
Rückgabe :  keine
Erklärung:  Diese Routine führt ein VSprite zum Gelsystem hinzu. Um
            ein VSprite erscheinen zu lassen, müssen die Copperlisten
            neu berechnet werden!
```

```
Routine   :  Animate (HeadOb,RastPort) (A0,A1)
Library   :  graphics.library
Offset    :  -162 = -$a2
Parameter:  A0 =  Zeiger auf einen Puffer von 4 Bytes (dc.l 0). Die-
                  ser Puffer zeigt immer auf das zuletzt eingefügte
                  `AnimOb`. (s.o.)
            A1 =  Zeiger auf `RastPort`-Struktur
Rückgabe :  keine
Erklärung:  Diese Routine animiert ein Objekt. Es berechnet entspre-
            chent die Werte in den Strukturen neu. Mit `SortGList ()`
            und `DrawGList ()` kann man dann das Objekt auf den
            Bildschirm darstellen.
```

Routine : AskSoftStyle (RastPort) (A1)
Library : graphics.library
Offset : -84 = -$54
Parameter: A1 = Zeiger auf RastPort-Struktur in dem der entsprechende Zeichensatz zu finden ist.
Rückgabe : in D0 = Stylebyte welches den aktuellen Modus des Fonts wiederspiegelt.

Stylename:	Wert:	Bit:	Funktion:
Normal	0	-	normal
Underline	1	0	unterstrichen
Bold	2	1	fett
Italic	4	2	kursiv
	8	3	extended

Erklärung: Diese Routine gibt in D0 den Style-Modus zurück, welcher gerade aktiv ist. Es sind auch mehrere Styls auf einmal möglich. Man braucht dazu lediglich die einzelnen Werte zusammenaddieren.

Funktion : CBump (UCopList) (A1)
Library : graphics.library
Offset : -366 = -$16e
Parameter: A1 = Zeiger auf `UCopList`-Struktur die angelegt wurde
Rückgabe : keine
Erklärung: Diese Routine erhöht den internen Zeiger der UserCopperliste. Muß nach jedem `CWait ()` und `CMove ()` aufgerufen werden!

Routine : ChangeSprite (ViewPort,SimpleSprite,Data) (A0,A1,A2)
Library : graphics.library
Offset : -420 = -$1a4
Parameter : A0 = Zeiger auf Viewport in dem der Sprite dargestellt werden soll.
A1 = Zeiger auf SimpleSprite-Struktur des Sprites

```
Routine  :  AddIntServer (Nummer,Interrupt) (D0,A1)
Library  :  exec.library
Offset   :  -168 = -$A8
Parameter:  D0 =  Interruptnummer - welcher IRQ benutzt werden soll
                  (Erlaubt sind: 3, 5, 13, 15)
            A1 =  Zeiger auf Interrupt-Struktur (siehe oben)
Rückgabe :  keine
Erklärung:  Diese Routine fügt einen Interrupt zum System hinzu
---------------------------------------------------------------

Routine  :  AddVSprite (VSprite,RastPort) (A0,A1)
Library  :  graphics.library
Offset   :  -102 = -$66
Parameter:  A0 =  Zeiger auf angelegte `VSprite`-Struktur
            A1 =  Zeiger auf `RastPort`-Struktur
Rückgabe :  keine
Erklärung:  Diese Routine führt ein VSprite zum Gelsystem hinzu. Um
            ein VSprite erscheinen zu lassen, müssen die Copperlisten
            neu berechnet werden!
---------------------------------------------------------------

Routine  :  Animate (HeadOb,RastPort) (A0,A1)
Library  :  graphics.library
Offset   :  -162 = -$a2
Parameter:  A0 =  Zeiger auf einen Puffer von 4 Bytes (dc.l 0). Die-
                  ser Puffer zeigt immer auf das zuletzt eingefügte
                  `AnimOb`. (s.o.)
            A1 =  Zeiger auf `RastPort`-Struktur
Rückgabe :  keine
Erklärung:  Diese Routine animiert ein Objekt. Es berechnet entspre-
            chent die Werte in den Strukturen neu. Mit `SortGList ()`
            und `DrawGList ()` kann man dann das Objekt auf den
            Bildschirm darstellen.
---------------------------------------------------------------
```

Routine : AskSoftStyle (RastPort) (A1)
Library : graphics.library
Offset : -84 = -$54
Parameter: A1 = Zeiger auf RastPort-Struktur in dem der entsprechende Zeichensatz zu finden ist.
Rückgabe : in D0 = Stylebyte welches den aktuellen Modus des Fonts wiederspiegelt.

Stylename:	**Wert:**	**Bit:**	**Funktion:**
Normal	0	-	normal
Underline	1	0	unterstrichen
Bold	2	1	fett
Italic	4	2	kursiv
	8	3	extended

Erklärung: Diese Routine gibt in D0 den Style-Modus zurück, welcher gerade aktiv ist. Es sind auch mehrere Styls auf einmal möglich. Man braucht dazu lediglich die einzelnen Werte zusammenaddieren.

Funktion : CBump (UCopList) (A1)
Library : graphics.library
Offset : -366 = -$16e
Parameter: A1 = Zeiger auf `UCopList`-Struktur die angelegt wurde
Rückgabe : keine
Erklärung: Diese Routine erhöht den internen Zeiger der UserCopperliste. Muß nach jedem `CWait ()` und `CMove ()` aufgerufen werden!

Routine : ChangeSprite (ViewPort,SimpleSprite,Data) (A0,A1,A2)
Library : graphics.library
Offset : -420 = -$1a4
Parameter : A0 = Zeiger auf Viewport in dem der Sprite dargestellt werden soll.
A1 = Zeiger auf SimpleSprite-Struktur des Sprites

```
              A2 =  Zeiger auf Spritedatas (Adresse steht in der Sim-
                    pleSprite-Struktur)
Rückgabe   : keine
Erklärung  : Diese Routine aktiviert einen Sprite auf dem Bildschirm.
---------------------------------------------------------------------

Routine   : CloseFont (TextFont) (A1)
Library   : graphics.library
Offset    : -78 = -$4e
Parameter:  A1 =  Zeiger auf eine TextFont-Struktur, den wird durch
                  die Routine "OpenDiskFont" in D0 erhalten haben
Rückgabe :  keine
Erklärung:  Diese Routine schließt einen Font wieder. Erst wenn ein
            Font geschlossen wurde, kann der Speicher, den er belegt
            wieder freigegeben werden.
---------------------------------------------------------------------

Routine   : CloseScreen (Screen) (A0)
Library   : intuition.library
Offset    : -66 = -$42
Parameter:  A0 =  Zeiger auf `Screen-Struktur`, welchen man in D0
                  durch die Routine `OpenScreen ()` erhält
Rückgabe :  keine
Erklärung:  Diese Routine schließt einen Screen auf den der Zeiger in
            A0 zeigt. Diesen Zeiger erhält man mit `OpenScreen ()`.
---------------------------------------------------------------------

Routine   : CloseWindow (Window) (A0)
Library   : intuition.library
Offset    : -72 = -$48
Parameter:  A0 =  Zeiger auf `Window-Struktur`, welchen man in D0
                  durch die Routine `OpenWindow ()` erhält
Rückgabe :  keine
Erklärung:  Diese Routine schließt ein Window auf den der Zeiger in
            A0 zeigt. Diesen Zeiger erhält man mit `OpenWindow ()`.
---------------------------------------------------------------------
```

Funktion : CMove (UCopList,Registernummer,Wert) (A1,D0,D1)
Library : graphics.library
Offset : -372 = -$174
Parameter: A1 = Zeiger auf `UCopList`-Struktur die angelegt wurde
D0 = Registernr. (z. B. #$180 für Hintergrundfarbregister) die Basisadresse wird nicht dabei übergeben (Basis = $Dff000)
D1 = Wert (z. B. #$00 für schwarze Farbe)
Rückgabe : keine
Erklärung: Diese Routine führt einen `Move`-Befehl in die angelegte UserCopperliste ein.

Funktion : CWait (UCopList,YPosition,XPosition) (A1,D0,D1)
Library : graphics.library
Offset : -378 = -$17a
Parameter: A1 = Zeiger auf `UCopList`-Struktur die angelegt wurde
D0 = Y-Position des Rasterstrahls auf die gewartet werden soll
D1 = X-Position des Rasterstrahls auf die gewartet werden soll nur in vierer Schritten möglich
Rückgabe : keine
Erklärung: Diese Routine führt einen `Wait`-Befehl in die angelegte UserCopperliste ein.

Routine : DoCollision (RastPort) (A1)
Library : graphics.library
Offset : -108 = -$6c
Parameter: A1 = Zeiger auf RasterPort-Struktur
Rückgabe : keine
Erklärung: Diese Routine testet ob eine Collision stattgefunden hat, und springt dann in die entsprechende Collisionsroutine. Bei BOB-BOB-Collisionen wird der angesprungenen Routine in A1 die Adresse des Oberen VSprites und in A2 die des unteren VSprites übergeben. Bei Randberührungen, wird der

angesprungenen Collisionsroutine (Nummer Null) in A3 die Adresse des VSprites übergeben.

Routine : DrawGList (RastPort,ViewPort) (A1,A0)
Library : graphics.library
Offset : -114 = -$72
Parameter: A0 = Zeiger auf `ViewPort`-Struktur
A1 = Zeiger auf `RastPort`-Struktur
Rückgabe : keine
Erklärung: Diese Routine zeichnet alle BOBs auf den Bildschirm bzw. in die Bitmap und generiert für alle VSprites eine neue Copperliste.

Routine : FreeColorMap (ColorMap) (A0)
Library : graphics.library
Offset : -576 = -$240
Parameter: A0 = Anfangsadresse der ColorMap-Struktur, die mit der Routine `GetColorMap ()` angelegt wurde
Rückgabe : keine
Erklärung: Diese Routine gibt den Speicher wieder frei, der mit `GetColorMap ()` für die `ColorMap`-Struktur angelegt wurde.

Routine : FreeCprList (CprList) (A0)
Library : graphics.library
Offset : -564 = -$234
Parameter: A0 = Anfangsadresse der LongFrame-Copperliststruktur, diese ist in der View-Struktur bei Offset 4 zu finden. Bei Hires muß auch ShortFrame-Copperliststruktur wieder freigegeben werden. (Offset 8)
Rückgabe : keine

Erklärung: Diese Routine gibt den Speicher wieder frei, der für die LongFrame-Copperliststruktur durch die Routine `MrgCop ()` vom System belegt wurde.

Routine : FreeSprite (Nummer) (D0)
Library : graphics.library
Offset : -414 = -$19e
Parameter: D0 = Spritenummer (von 0 bis 7)
Rückgabe : keine
Erklärung: Diese Routine gibt ein reserviertes Sprite vom System wieder frei.

Routine : FreeVPortCopLists (ViewPort) (A0)
Library : graphics.library
Offset : -540 = -$21c
Parameter: A0 = Anfangsadresse der `ViewPort`-Struktur
Rückgabe : keine
Erklärung: Diese Routine gibt den Speicher wieder frei, der für die einzelnen Copperlisten durch die Routine `MakeVPort ()` vom System belegt wurde.

Routine : GetColorMap (entries) (D0)
Library : graphics.library
Offset : -570 = -$23a
Parameter: D0 = Anzahl Farben (max. 32 Farben)
Rückgabe : in D0 = Anfangsadresse der vom System angelegten `Color-Map`-Struktur
Erklärung: Diese Routine erzeugt eine ColorMap-Struktur, deren Adresse in D0 zurückgegeben wird. Wenn eine Null zurückgegeben wird, dann konnte nicht genug Speicher für diese Routine angelegt werden. Diese vom System angelegte Struktur muß man dann nur noch mit der `View-Port`-Struktur verbinden (Offset 4).

Routine : GetRGB4 (ColorMap,entry) (A0,D0)
Library : graphics.library
Offset : -582 = -$246
Parameter: A0 = Anfangsadresse der zu diesem Display gehörenden ColorMap-Struktur
D0 = Farbregisternummer, aus dem der Farbwert entnommen werden soll (0-31)
Rückgabe : in D0 = Farbwert des angegebenen Farbregisters
Erklärung: Diese Routine gibt den Farbwert des gewünschten Farbregisters in D0 zurück.

Routine : GetSprite (SimpleSprite,Nummer) (A0,D0)
Library : graphics.library
Offset : -408 = -$198
Parameter: A0 = Zeiger auf SimpleSprite-Struktur des Sprites
D0 = Spritenummer (von 0 bis 7)
Rückgabe : D0 = Spritenummer welcher vorher übergeben wurde
= -1 für Sprite konnte nicht reserviert werden
Erklärung: Diese Routine reserviert einen Sprite, welcher dann manipuliert werden kann.

Routine : InitBitMap (BitMap,depth,width,heigth) (A0,D0,D1,D2)
Library : graphics.library
Offset : -390 = -$186
Parameter: A0 = Anfangsadresse der angelegten `BitMap`-Struktur
D0 = Anzahl Bitplanes (Tiefe)
D1 = Breite der Bitmap
D2 = Höhe der Bitmap
Rückgabe : keine
Erklärung: Diese Routine initialisiert die angelegte `BitMap`-Struktur mit den wichtigsten Werten. Die PlanePtrX-Adressen müssen dann nur noch in der BitMap-Struktur von Ihnen selbst eingetragen werden.

```
Routine  :  InitGels (GelHead,GelTail,GelsInfo) (A0,A1,A2)
Library  :  graphics.library
Offset   :  -120 = -$78
Parameter:  A0 =  Zeiger auf einen Puffer mit 58 Null-Bytes
            A1 =  Zeiger auf einen Puffer mit 58 Null-Bytes
            A2 =  Zeiger auf angelete `GelsInfo`-Struktur
Rückgabe :  keine
Erklärung:  Diese Routine initialisiert das Gelsystem. Die `GelsIn-
            fo`- Struktur braucht dann nur noch mit der `Rast-
            Port`-Struktur verbunden werden (Offset 20)!
-------------------------------------------------------------------

Routine  :  InitMasks (VSprite) (A0)
Library  :  graphics.library
Offset   :  -126 = -$7e
Parameter:  A0 = Zeiger auf angelegte `VSprite`-Struktur
Rückgabe :  keine
Erklärung:  Diese Routine initialisiert die Variablen `CollMask`,
            `BorderLine`, und `ImageShadow`. Es muß natürlich vorher
            genügend Speicher reserviert worden sein.
-------------------------------------------------------------------

Routine  :  InitRastPort (RastPort) (A1)
Library  :  graphics.library
Offset   :  -198 = -$c6
Parameter:  A1 =  Anfangsadresse der `RastPort`-Struktur die man an-
                  gelegt hat
Rückgabe :  keine
Erklärung:  Diese Routine initialisiert die angelegte `Rast-
            Port`-Struktur mit den wichtigsten Werten
-------------------------------------------------------------------

Routine  :  InitView (View) (A1)
Library  :  graphics.library
Offset   :  -360 = -$168
Parameter:  A1 = Anfangsadresse der angelegten `View`-Struktur
```

Rückgabe : keine
Erklärung: Diese Routine initialisiert die angelegte `View`-Struktur mit den wichtigsten Werten

---

Routine : InitVPort (ViewPort) (A0)
Library : graphics.library
Offset : -204 = -$cc
Parameter: A0 = Anfangsadresse der angelegten `ViewPort`-Struktur
Rückgabe : keine
Erklärung: Diese Routine initialisiert die angelegte `View-Port`-Struktur mit den wichtigsten Werten

---

Routine : LoadRGB4 (ViewPort,Colors,Count) (A0,A1,D0)
Library : graphics.library
Offset : -192 = -$c0
Parameter: A0 = Anfangsadresse der zu diesem Display gehörenden ViewPort-Struktur
A1 = Anfangsadresse vom einem 32 Word großen Array, in denen sich die Farbwerte befinden
D0 = Anzahl der Farbregister, welche mit neuen Farben gefüllt werden soll
Rückgabe : keine
Erklärung: Diese Routine lädt in die ColorMap des angegebenen View-Ports die gewünschten Farbwerte, welche sich in dem Array befinden. Es müssen danach aber erst die Copperlisten neu berechnet werden, damit der Farbwechsel sichtbar wird.

Routine : LoadView (View) (A1)
Library : graphics.library
Offset : -222 = -$de
Parameter: A1 = Anfangsadresse der `View`-Struktur
Rückgabe : keine

Erklärung: Diese Routine startet die Copperliste, auf die die Long-Frame-Copperlist-Struktur zeigt, über die Hardware. Und erzeugt somit das fertige Bild.

Routine : MakeScreen (Screen) (A0)
Library : intuition.library
Offset : -378 = -$17a
Parameter: A0 = Zeiger auf `Screen`-Struktur
Rückgabe : keine
Erklärung: Diese Routine führt die Funktion `MakeVPort ()` für den ViewPort des angegebenen Screen durch.

Routine : MakeVPort (View,ViewPort) (A0,A1)
Library : graphics.library
Offset : -216 = -$d8
Parameter: A0 = Anfangsadresse der `View`-Struktur
A1 = Anfangsadresse der `ViewPort`-Struktur
Rückgabe : keine
Erklärung: Diese Routine erzeugt die einzelnen Copperliststrukturen und setzt diese in die entsprechenden Zeiger in der `ViewPort`-Struktur ein. Vorher muß natürlich die View-Port-Struktur mit der View-Struktur verbunden werden.

Routine : Move (RastPort,X,Y) (A1,D0,D1)
Library : graphics.library
Offset : -240 = -$f0
A1 = Zeiger auf RastPort-Struktur
D0 = Position der X-Koordinate
D1 = Position der Y-Koordinate
Erklärung: Diese Routine setzt die Koordinaten im entsprechenden RastPort. Ab diesen Positionen wird dann der Text bzw. die Linie dargestellt.

```
Routine  :  MoveSprite (ViewPort,SimpleSprite,X,Y) (A0,A1,D0,D1)
Library  :  graphics.library
Offset   :  -426 = -$1aa
Parameter:  A0 =  Zeiger auf Viewport in dem der Sprite dargestellt
                  werden soll.
            A1 =  Zeiger auf SimpleSprite-Struktur des Sprites
            D0 =  X-Position an der der Sprite erscheinen soll
            D1 =  Y-Position an der der Sprite erscheinen soll
Rückgabe :  keine
Erklärung:  Diese Routine läßt den Sprite auf die angegebenen Koor-
            dinaten erscheinen.
---------------------------------------------------------------------

Routine  :  MrgCop (View) (A1)
Library  :  graphics.library
Offset   :  -210 = -$d2
Parameter:  A1 = Anfangsadresse der `View`-Struktur
Rückgabe :  keine
Erklärung:  Diese Routine erzeugt aus den einzelnen Copperliststruk-
            turen die mit `MakeVPort ()` erzeugt wurden eine einzige
            Copperlist-Struktur. Diese wird dann in die View-Struktur
            eingehängt (LongFrame-Copperliststruktur)
---------------------------------------------------------------------

Routine  :  OpenDiskFont (TextAttr) (A0)
Library  :  diskfont.library (befindet sich auf Disk - Workbench)
Offset   :  -30 = -$1e
Parameter:  A0 =  Zeiger auf eine Text-Attribut-Struktur, welche den
                  gewünschten Font genauer beschreibt. (siehe unten)
Rückgabe :  D0 =  Zeiger auf TextFont-Struktur (siehe Aufbau der
                  Fonts), die zu diesem Font gehört; oder Null wenn
                  Error aufgetreten ist.
Erklärung:  Diese Routine lädt einen Font von Disk in den Speicher
            und legt die dazugehörige TextFont-Struktur an.
---------------------------------------------------------------------
```

Rückgabe : keine
Erklärung: Diese Routine entfernt einen Interrupt wieder aus dem System

---

Routine : RemVSprite (VSprite) (A0)
Library : graphics.library
Offset : -138 = -$8a
Parameter: A0 = Zeiger auf angelegte \`VSprite\`-Struktur
Rückgabe : keine
Erklärung: Diese Routine entfernt ein VSprite für immer aus dem Gelsystem!

Routine : RethinkDisplay ()
Library : intuition.library
Offset : -390 = -$186
Parameter: keine
Rückgabe : keine
Erklärung: Ruft für alle Screens \`MrgCop ()\` und \`LoadView ()\` auf

Routine : ScrollRaster (RastPort,DX,DY,MinX,MinY,MaxX,MaxY) (A1,D0 bis D5)
Library : graphics.library
Offset : -396 = -$18c
Parameter: A1 = Zeiger auf RastPort-Struktur
D0 = Anzahl X-Punkte um die gescrollt werden soll
D1 = Anzhal Y-Punkte um die gescrollt werden soll
D2 = Anfangs X-Koordinate
D3 = Anfangs Y-Koordinate
D4 = End X-Koordinate
D5 = End Y-Koordinate
Erklärung: Diese Routine scrollt den angegebenen Bereich um die entsprechende Anzahl von Punkten

---

Routine : SetAPen (RastPort,Pen) (A1,D0)
Library : graphics.library
Offset : -342 = -$156
Parameter: A1 = Zeiger auf RastPort-Struktur
D0 = Farbregisternummer, aus dem die Schriftfarbe entnommem werden soll.
Erklärung: Diese Routine setzt die Schriftfarbe im entsprechenden RastPort.

Routine : SetBPen (RastPort,Pen) (A1,D0)
Library : graphics.library
Offset : -348 = -$15c
Parameter: A1 = Zeiger auf RastPort-Struktur
D0 = Farbregisternummer, aus dem die Hintergrundfarbe entnommem werden soll.
Erklärung: Diese Routine setzt die Hintergrundfarbe für die Textausgabe im entsprechenden RastPort.

Routine : SetCollision (Type,Routine,GelsInfo) (D0,A0,A1)
Library : graphics.library
Offset : -144 = -$90
Parameter: D0 = Nummer der Collisionsroutine (0-15)
A0 = Zeiger auf Collisionsroutine die mit einem RTS endet
A1 = Zeiger auf GelsInfo-Struktur
Rückgabe : keine
Erklärung: Diese Routine init. eine Collsionsroutine mit angegebener Nummer. Die Adresse der Colisionsroutine wird in den `CollHandler`- Puffer eingetragen.

Routine : SetDrMd (RastPort,DrawMode) (A1,D0)
Library : graphics.library
Offset : -354 = -$162
Parameter: A1 = Zeiger auf RastPort-Struktur
D0 = Zeichenmodus für die Textausgabe:

Name:	**Bit:**	**Funktion:**
JAM1	-	Es wird nur in der Farbe des APens gemalt
JAM2	0	Es wird mit der Farbe des BPens unterlegt
Complement	1	Nur gesetzte Punkte werden in Screen ausgegeben
Inversvid	2	Text wird invertiert ausgegeben

Erklärung: Diese Routine setzt den Zeichenmodus für die Textausgabe im entsprechenden RastPort.

Routine : SetFont (RastPort,TextFont) (A1,A0)
Library : graphics.library
Offset : -66 = -$42
Parameter: A1 = Zeiger auf eine Rasterportstruktur in dem der gewünschte Font aktiviert werden soll.
A0 = Zeiger auf eine TextFont-Struktur, den wird durch die Routine "OpenDiskFont" in D0 erhalten haben
Rückgabe : keine
Erklärung: Diese Routine aktiviert einen Font im entsprechenden Rasterport.

Routine : SetRast (RastPort,Color) (A1,D0)
Library : graphics.library
Offset : -234 = -$ea
Parameter: A1 = Zeiger auf RastPort-Struktur in dem der entsprechende Zeichensatz zu finden ist.

D0 = Farbregisternummer, aus dem die Farbe entnommen werden soll, fürs zeichnen von Linien oder ausgeben von Texten

Erklärung: Diese Routine setzt die Punktfarbe im entsprechenden RastPort.

Routine : SetRGB4 (ViewPort, Register, rot, gelb, blau) (A0, D0, D1, D2, D3)

Library : graphics.library

Offset : -288 = -$120

Parameter: A0 = Anfangsadresse der zu diesem Display gehörenden ViewPort-Struktur

D0 = Farbregisternummer, in dem die Farbänderung vorgenommen werden soll (0-31)

D1 = Rotanteil der Farbe (Wert von 0-15)

D2 = Gelbanteil der Farbe (Wert von 0-15)

D3 = Blauanteil der Farbe (Wert von 0-15)

Rückgabe : keine

Erklärung: Diese Routine lädt das angegebene Farbregister des entsprechenden ViewPorts mit dem gewünschten Wert. Die Farbänderung wird sofort sichtbar, die Copperlisten brauchen also nicht neu berechnet werden.

Routine : SetSoftStyle (RastPort,Style,Enable) (A1,D0,D1)

Library : graphics.library

Offset : -90 = -$5a

Parameter: A1 = Zeiger auf RastPort-Struktur in dem der entsprechende Zeichensatz zu finden ist.

D0 = Stylebyte welches den aktuellen Modus des Fonts wiederspiegelt.

Stylename:	**Wert:**	**Bit:**	**Funktion:**
Normal	0	-	normal
Underline	1	0	unterstrichen

Bold	2	1	fett
Italic	4	2	kursiv
	8	3	extended

D1 = Enthält den Wert, der Auskunft darüber gibt, welche Styles überhaupt zugelassen sind. Also die Stylemaske. Am besten Sie setzen D1 auf 15.

Erklärung: Diese Routine setzt den Style-Modus im entsprechenden RastPort. Wenn Sie jetzt einen Text in diesem RastPort ausgeben, wird dieser entsprechend anders ausgegeben.

Routine : SortGList (RastPort) (A1)
Library : graphics.library
Offset : -150 = -$96
Parameter: A1 = Zeiger auf `RastPort`-Struktur
Rückgabe : keine
Erklärung: Diese Routine sortiert das Gelsystem (Position etc.). Muß vor jedem `DrawGList ()` aufgerufen werden!

Routine : Text (RastPort,String,Count) (A1,A0,D0)
Library : graphics.library
Offset : -60 = -$3c
Parameter: A1 = Zeiger auf RastPort-Struktur in dem der Text dargestellt werden soll.
A0 = Anfangsadresse des auszugebenen Textes.
D0 = Anzahl Buchstaben die ausgegeben werden sollen.
Rückgabe : keine
Erklärung: Diese Routine gibt eine Anzahl von Buchstaben auf dem Bildschirm aus. Allerdings an den aktuellen Koordinaten bzw. in der aktuellen Schriftfarbe.

Routine : TextLength (RastPort,String,Count) (A1,A0,D0)
Library : graphics.library
Offset : -54 = -$36
Parameter: A1 = Zeiger auf RastPort-Struktur in dem der entsprechende Zeichensatz zu finden ist.
A0 = Anfangsadresse des Textes.
D0 = Anzahl Buchstaben die der Text besitzt
Rückgabe : in D0 = Anzahl Punkte die der Text lang ist.
Erklärung: Diese Routine gibt in D0 die Anzahl Punkte zurück, die der Text lang ist. Die Textlänge bezieht sich immer auf die aktuelle Zeichenbreite des aktiven Zeichensatzes im übergebenen RastPort.

Routine : ViewAdress ()
Library : intuition.library
Offset : -294 = -$126
Parameter: keine
Rückgabe : in D0 = Anfangsadresse der aktuellen `View`-Struktur
Erklärung: Diese Routine gibt in D0 einen Zeiger auf die aktive `View`-Struktur, die vom System verwaltet wird, zurück.

Routine : ViewPortAdress (Window) (A0)
Library : intuition.library
Offset : -300 = -$12c
Parameter: A0 = Zeiger auf `Window`-Struktur
Rückgabe : in D0 = Anfangsadresse der aktuellen `ViewPort`-Struktur
Erklärung: Diese Routine gibt in D0 einen Zeiger auf die zu diesem Window gehörende `ViewPort`-Struktur zurück.

Anhang C - Die Programme der Diskette

Auf der beiliegenden Diskette befinden sich alle 19 Programme die im Buch enthalten sind. Einmal als lauffähiges Programm und jeweils das dazugehörige Source-Listing.

Im Verzeichnis 'Programme' befinden sich folgende laufähige Programme (Durch aufrufen des Namens werden sie gestartet):

AnimateBOB
BitMap
BOB-Collision
Brush-Transformer
Copper
Display
Dualplayfield
Hires
IFF-Loader
Laufschrift
MoveBOB
OpenFont
RasterIRQ
RasterIRQ(Server)
Screen
Scrolling
SimpleSprite
SoftIRQ
VSprites

Im Verzeichnis 'Listings' befinden sich alle Source-Codes, unter dem selben Namen wie oben aufgeführt.

Stichwortverzeichnis

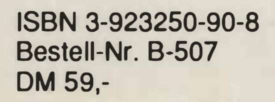
Grafik in Assembler auf dem Amiga

WICHTIGE MERKMALE:

Zahlreiche Beispielprogramme sorgen dafür, daß das Erlernte nicht nur Theorie bleibt. Dabei wird auch auf Scrolling, HAM-Modus, Dual Play Field, Copper-Programmierung, Fonts laden und anzeigen, Simple Sprites erzeugen, eingegangen. Auch das komplette Animationssystem wird beschrieben, darunter V-Sprites, Bobs, doppeltgepufferte Bobs, animierte Bobs, Collision-Abfrage und andere. Ferner die Interrupt-Programmierung, die Joystick-Abfrage in 16 Richtungen und die Erzeugung von Laufschriften. Schließlich erlaubt der IFF-Standard, Bilder und Brushes in Ihre eigenen Programme einzubauen.

AUS DEM INHALT:

Grafikmodi:
  Hold and Modify (4096 Farben) • Hires • Dual Play Field • Scrolling
Copper:
  User Copper Liste • Copper Routinen des Systems
Programmierung unter Intuition:
  Screens öffnen/schließen • Fenster öffnen/schließen
Interrupts:
  User IRQ • Raster IRQ
Fonts:
  Aufbau von Fonts • Texte ausgeben • Laufschriften
Joystick-Abfrage:
  Tastatur-Abfrage
Simple Sprites:
  Erzeugung und Aufbau
Das Animationssystem:
  V-Sprites • Bobs • Bob-Routinen • Animation • Collision
IFF Standard:
  Screens • Brushes


ISBN 3-923250-90-8
Bestell-Nr. B-507
DM 59,-

